Crie um instalador de componentes: Facilitando a instalação de componentes no Delphi

Clube Delphi

Explorando APIs do Windows
Trabalhe com o mouse, impressoras e áudio

Desenvolvendo um Sistema Financeiro
Criando as camadas finais da aplicação

Edição 165 :: R$ 14,90

DEVMEDIA

ISSN 1517990-7

MVP

R$ 1.000.000,00
INVESTIDOS EM CONTEÚDO NOS ÚLTIMOS 12 MESES.

APLIQUE ESSE INVESTIMENTO NA SUA CARREIRA...

E MOSTRE AO MERCADO QUANTO VOCÊ VALE!

CONFIRA TODO O MATERIAL QUE VOCÊ TERÁ ACESSO:

- + de **9.000** video-aulas
- + de **290** cursos online
- + de **13.000** artigos
- DEVMEDIA API's consumido + de **500.000** vezes

POR APENAS
R$ 69,90* mensais

*Tempo mínimo de assinatura: 12 meses.

ASSINATURA MVP MOST VALUABLE PROFESSIONAL

PRA QUEM QUER EXIGIR
MAIS DO MERCADO!

DEVMEDIA

Clube Delphi

165ª Edição 2015 ISSN 1517990-7 Impresso no Brasil

Editor Geral

Joel Rodrigues (joelrlneto@gmail.com)

Equipe Editorial

Guinther Pauli (guintherpauli@gmail.com)

Fabrício Hissao Kawata (fabricio.kawata@bol.com.br)

Giuliano Scombatti Pinto (giuliano@sygnux.com.br)

Daniel Sobrinho Laporte (daniel.laporte@gmail.com)

Jornalista Responsável

Kaline Dolabella - JP24185

Consultora Técnica

Daniella Costa (daniella.devmedia@gmail.com)

Capa e Diagramação

Romulo Araujo

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Joel Rodrigues - Editor da Revista
joelrlneto@gmail.com

Joel Rodrigues - **Editor Geral**

Editor da ClubeDelphi Magazine

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DEVMEDIA

Sumário

Conteúdo sobre Novidade

04 – Conheça a Rad Studio 10 Seattle
[*Rodrigo Carreiro Mourão*]

Artigo no estilo Série

18 – Explorando APIs do Windows em Delphi – Parte 2
[*Vanderson Cavalcante de Freitas*]

Artigo no estilo Curso

28 – Desenvolvendo um Sistema Financeiro em Delphi – Parte 4
[*Filipe Dalepiane*]

Artigo no estilo Solução Completa

40 – Como criar um instalador de componentes
[*Gutierry Antoniow*]

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Conheça a Rad Studio 10 Seattle

Tudo sobre a nova versão do Delphi

O Rad Studio é a ferramenta de desenvolvimento multiplataforma da Embarcadero Technologies, com ela é possível desenvolver aplicações nativas para as plataformas Win32, Win64, Android, iOS32, iOS64 e OSX a partir de um único código fonte. Nós escrevemos o código uma única vez e compilamos para todas as plataformas citadas. Isso é possível graças a presença de sete compiladores que fazem o trabalho pesado para nós.

No final de agosto de 2015 a Embarcadero lançou oficialmente a mais nova versão do Rad Studio, o Rad Studio 10 Seattle. Essa nova versão também é conhecida como Delphi 10, DX ou Delphi Seattle. O Rad Studio é na verdade a suíte de desenvolvimento que é composta pelo Delphi e C++, assim o RX é formado pelo DX e o CX. Quando falamos em Rad Studio estamos falando de Delphi e C++ e é importante enfatizar isso, pois há funcionalidades que estão presentes no Rad Studio e funcionam tanto para linguagem Delphi quando C++, geralmente funcionalidade do IDE, e há também novos recursos que são exclusivos para uma ou outra linguagem. Assim, neste artigo falaremos das novidades que dizem respeito ao Delphi Seattle.

Mesmo com a funcionalidade de desenvolvimento cross-plataform (**BOX 1**) presente no Rad Studio há alguns anos, é fato que o Delphi ainda é muito forte no desenvolvimento de aplicações para plataforma Windows e o foco desta nova versão é exatamente este, tendo isso um propósito que falaremos mais à frente.

A Embarcadero continua mantendo a média de uma nova versão a cada oito meses. Esse procedimento tem o lado bom e o lado ruim. O lado bom é que a ferramenta evolui muito rápido, novas features são incluídas e o produto acompanha a evolução da tecnologia. O lado ruim é que não dá para acompanhar e se aprofundar em tudo de novo que é lançado. Há também uma polêmica por conta de quem adquire uma licença e logo vê seu produto se tornar obsoleto. Para estes a Embarcadero possui um modelo de licenciamento de assinatura que garante a atualização das versões sem ter que pagar a mais por isso. Para saber mais sobre como funciona a atualização de acordo com a licença do Rad Studio, veja a seção **Links** no final do artigo.

Fique por Dentro

Recentemente a Embarcadero lançou a nova versão do Delphi, o Delphi 10 Seattle. Neste artigo faremos uma viagem pela história do Delphi até chegar na versão atual para termos uma ideia do que rolou até aqui. Além disso, vermos as novidades do IDE que nos permitem produzir mais com menos e as novidades da VCL. Como esta nova versão tem como foco o desenvolvimento para Windows 10 faremos alguns exemplos com os novos componentes exclusivos para esta plataforma.

BOX 1. Cross-Plataform

Cross-Plataform é uma técnica para desenvolver uma aplicação a partir de uma única base de código e permitir que esta possa ser utilizada em diferentes sistemas operacionais, seja desktop ou mobile. Este tipo de abordagem é vantajoso, já que não é necessário duplicar o código da aplicação para cada plataforma desejada, muito menos manter equipes distintas e especializadas em cada uma das plataformas. Logo, o desenvolvimento cross-plataform reduz o custo e o prazo de desenvolvimento da aplicação.

Windows 10

O lançamento do Windows 10 marca uma nova era para a Microsoft, que tem uma meta ambiciosa de atingir a marca de um bilhão de dispositivos rodando Windows dentro de dois ou três anos. Essa nova fase na Microsoft lança um novo conceito de sistema operacional, o Windows as Service (ou Windows como serviço).

O Delphi vem mantendo a tradição de sair na frente e isso desde a época do Windows 95. Foi assim com desenvolvimento 16 bits, depois suporte a 32 bits, Threads, Vista Dialogs, Touch e Gestures, Interface Metro e não está sendo diferente com o RX. A VCL incorpora novos componentes para que de forma transparente o desenvolvedor possa acessar a API do Windows 10 e utilizar os novos recursos, um exemplo disso é o novo componente TNotificationCenter que permite enviar notificações para um dispositivo rodando Windows 10.

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Histórico das versões

Antes de entramos no assunto da versão atual, é importante fazermos uma retrospectiva para que possamos nos contextualizar sobre a evolução e estágio atual da ferramenta. A seguir veremos alguns dos principais tópicos lançados em cada uma das versões do Delphi até agora.

Na **Tabela 1** vemos a evolução do Delphi 1 ao Delphi 6, foram os anos onde a ferramenta se destacou no cenário mundial como ferramenta de desenvolvimento para a Windows. Em 2001 foi lançada a versão 7 do Delphi, a última versão estável antes do lançamento do conceito de "Studio" pela Borland, conforme mostra a evolução descrita na **Tabela 2**.

De 2007 a 2009 tivemos dois grandes movimentos que foram fundamentais para o destino do Delphi. Primeiro o desmembramento da Borland, que decidiu focar em ALM e criar uma subsidiária para cuidar das ferramentas de desenvolvimento, surgia ali a Code Gear. Em 2008 a Code Gear foi comprada pela Embarcadero e já em 2009 lançou o Embarcadero Rad Studio 2010, conforme mostra a trajetória descrita na **Tabela 3**.

Nos últimos cinco anos foram lançadas nada menos que oito versões, com o Delphi chegando em sua versão 22, que recebeu o nome de Delphi XE8 e com ele o ciclo de versões batizadas com o sufixo XE chega ao fim. Veja na **Tabela 4** as principais funcionalidades adicionadas em cada uma das referidas versões.

Na seção **Links** encontra-se um post da Embarcadero contando a história do Delphi em seus 20 anos de existência.

O Rad Studio 10

O nome dado à versão é bem sugestivo, já que também recentemente a Microsoft lançou a mais nova versão de seu sistema operacional, o Windows 10, este é o principal motivo que fez a Embarcadero optar por não lançar o Delphi XE9, mas sim pular (ao menos no nome) uma versão e partir para a versão 10. O número 10 em algarismo Romano também remete à antiga nomenclatura já que é representado pela letra X. O Delphi manterá a nomenclatura 10 em seu nome até segunda ordem, isso significa que a próxima versão não será Delphi 11, o número 10 passou a fazer parte do nome da ferramenta, assim como ocorrerá com o Windows. O que vai mudar de uma versão para a outra é o nome da cidade que virá junto com o nome do produto, a cada nova versão uma cidade diferente será homenageada. Nesta versão a cidade escolhida foi Seattle, provavelmente pelo fato da nova versão do Delphi estar focada no desenvolvimento Windows e a sede da Microsoft estar instalada nessa cidade.

Novidades no IDE

A maior das novidades do IDE desta versão não é um novo menu, uma nova feature ou um novo refactor, mas sim um novo IDE.

Ano	Versão	SO	Características
1995	1.0	Win 16	Integração com Windows 16bit
1996	2.0	Win 32	Suporte total ao Windows 95 (32bit) e integração com OLE automation, além da herança visual de formulários
1997	3.0	Win 32	Surgimento do Code Insight e Packages
1998	4.0	Win 32	Suporte ao Windows 98 e melhora do OLE e COM
1999	5.0	Win 32	Layouts de desktop e melhorias no depurador
2000	6.0	Win 32 Linux	Suporte ao desenvolvimento Cross-platform, Web Services, dbExpress e IntraWeb

Tabela 1. Evolução da versão do Delphi 1 ao Delphi 6

Ano	Versão	SO	Características
2001	7.0	Win 32 .NET	Delphi for .NET Framework Preview; compilador de linha de comando; melhorias na IDE com HTML e XML; melhorias no suporte a SOAP e Web Services, ModelMaker e Rave Report
2003	8.0	.NET	Integração total ao .NET Framework, ECO, ASP.NET e ADO.NET
2004	2005 (9)	Win 32 .NET	Suporte a múltiplas personalidades de linguagem, Refactoring, ECO II, ADO.NET Remoting e integração com StarTeam
2005	BDS 2006 (10)	Win 32 .NET	Suporte para Delphi, C, C++ e C#, ECO III; modelagem UML com Together LiveSource; auditorias e métricas e geração de Documentação; ECO III Model Driven Development

Tabela 2. Evolução do Delphi 7 ao BDS 2005

Ano	Versão	SO	Características
2007	Rad Studio 2007 (11)	Win32 .Net	Suporte a .Net 2.0; ECOIV; melhorias no compilador e correção em mais de 100 bugs; DBExpress 4; MSBuild; novos componentes na VCL e classes na RTL; suporte a Windows Vista
2008	CodeGear Delphi 2009 (12)	Win32	.NET no Delphi Prism; Ribbon Control; Generics; Anonymous Methods; suporte a Unicode; novo framework DataSnap
2009	Embarcadero Delphi 2010 (14)	Win32	Melhorias no Unicode; nova RTTI; attributes, Win7; Direct2D; suporte a Touch, Multi-Touch e Gestures; novas classes DataSnap com suporte a REST; acesso a Firebird.

Tabela 3. Evolução do Rad Studio 2007 ao Embarcadero Rad Studio 2010

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Ano	Versão	SO	Características
2010	Embarcadero DelphiXE (15)	Win32	Novas classes e wizards para DataSnap; AQTime, CodeSite, FinalBuilder e SubVersion; Windows Azure e Amazon EC2; atualizações na VCL e RTL, além de melhorias no compilador
2011	Embarcadero DelphiXE2 (16)	Win32 Win64 OSx	Suporte a compilação nativa em 32bits e 64bits; FireMonkey para aplicações mais ricas; desenvolvimento para MacOs e iOS; aplicações em 3D; melhorias no compilador, debbuger, RTTI e DBExpress; DataSnap Mobile Connectors (Android, Blackberry, iOS e WinPhone) e LiveBindings
2012	Embarcadero DelphiXE3 (17)	Win32 Win64 OSx	Nova Interface Metro Windows 8 com novos Wizard para aplicações VCL e FMX; melhorias no FireMonkey, que agora suporta Action e Gestures; adicionado componentes Sensores; Livebindings completamente reformulado e driver SQL Lite para Windows e Mac
2013 (Abr)	Embarcadero DelphiXE4 (18)	Win32 Win64 OSx iOS	Novo FireMonkey com suporte ao desenvolvimento mobile para iOS; dezenas de novos componentes e funções; dois novos compiladores DCCIOS32.EXE e DCCIOSARM.EXE; Refatoramento de várias units; Mobile Form Designer
2013 (Set)	Embarcadero DelphiXE5 (19)	Win32 Win64 MacOs iOS Android	Suporte a desenvolvimento para Android; melhorias no TListView FMX; suporte à arquivo de áudio iOS e Android; novas configurações em Tools Options e Project Options; FireDac; novas classes RTL e novos Refactories em units FMX
2014 (Abr)	Embarcadero DelphiXE6 (20)	Win32 Win64 MacOs iOS Android	Melhorias no FireMonkey como suporte a Osx Lion 10.7; novos Styles para Google Glass; novos recursos e melhorias na API FMX; App Tethering; styles para VCL App; melhorias na IDE com novos ícones, além de novas opções de deploy, tools e Project Options com novas opções; melhorias Firedac (Push iOS) e Directx11 OpenGL 4.3
2014 (Set)	Embarcadero DelphiXE7 (21)	Win32 Win64 MacOs iOS Android	FireMonkey passa a ser Multi-Device Application; TMultiView Component; melhorias no Firedac; EMS; BDE Removida da IDE; dezenas de melhorias e novos recursos no FMX; suporte a GIT; RAD Studio Guided Tours
2015	Embarcadero DelphiXE8 (22)	Win32 Win64 MacOs iOS Android	Suporte a Beacons e Bluetooth LE; compilação iOS64; novo Multi-Device Preview; novos componentes de reenderização nativa no iOS; Embarcadero Community Toolbar; GetIt Package Manager; novo Styles e melhorias na IDE, RTL e FireDac.

Tabela 4. Evolução da família XE

O IDE foi redefinido e agora utiliza o dobro de memória que a versão anterior. Na prática isso significa dizer que o Delphi agora pode trabalhar com projetos grandes sem "engasgar", como aqueles que são agrupados em um grupo de projetos com dezenas de subprojetos, módulos em pacotes, classes de persistência e centenas de milhares de linha de código.

Além disso o IDE incorpora agora um mecanismo de recuperação de crash, assim caso o IDE por algum motivo se encerre de forma inesperada, o seu projeto é restaurado ao estado anterior ao fechamento acidental, tudo isso para evitar perder o que estávamos fazendo.

O structure panel, aquele panel localizado na parte superior esquerda do IDE, agora exibe um ícone em seus elementos que representa fielmente o componente que está sendo representado. Isso facilita muito na interpretação visual da estrutura de componentes, já que muitos dos componentes colocados em tela não são renomeados e quando são podem não seguir um padrão de nomenclatura que indique o tipo de objeto que aquele elemento é. Veja na **Figura 1** um exemplo do structure mostrando o elemento de um projeto VCL, note que a presença dos ícones de fato facilita a leitura visual dos elementos dispostos no formulário.

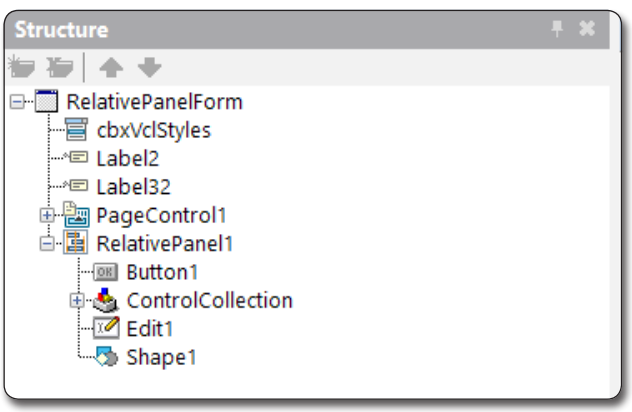


Figura 1. Structure Panel

O Object Inspector, um velho conhecido dos programadores, também melhorou. Ele recebeu um elemento filtrante, agora podemos pesquisar as propriedade e eventos de um objeto qualquer apenas digitando parte de seu nome na caixa de pesquisa. Pode não parecer, mas isso aumenta a produtividade, já que com o passar do tempo os componentes receberam dezenas de novas propriedade e navegar por elas em um determinado componente

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

poderia ser penoso. A **Figura 2** mostra o filtro do Object Inspector em ação.

Outro recurso interessante adicionado a esta nova versão é a possibilidade de ocultar dos formulários os componentes não visuais. Todos sabem que eles possuem este nome pois não são exibidos no projeto em tempo de execução, porém em tempo de desenvolvimento eles são visíveis. Agora através do atalho Ctrl + H podemos ocultar ou exibir os componentes não visuais do projeto. Esta funcionalidade também pode ser acessada pela barra de tarefas da IDE ou pelo menu Edit -> Hide Non-Visual Components. Veja na **Figura 3** a diferença de um form exibindo seus controles não visuais e o mesmo form com estes controles ocultos.

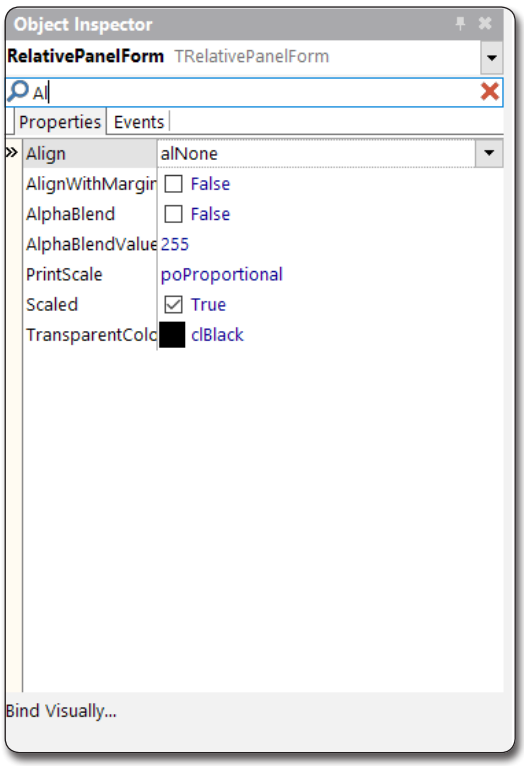


Figura 2. Object Inspector

Ainda no IDE temos alguns outros recursos que embora não tenham sido adicionados na versão Seattle, ainda são bem recentes, pois foram inseridos na versão XE 8 que rapidamente foi substituída. Para começar, a nova Embarcadero Community Toolbar, que nos dá acesso a algumas funcionalidades da comunidade de desenvolvedores da Embarcadero. A antiga EDN (Embarcadero Developer Network) está sendo migrada para a nova Community e através deste toolbar é possível efetuar login e postar dúvidas, acessar o calendário de eventos e trocar mensagens com os demais membros. Na **Figura 4** é possível ver a Community em ação exibindo dentro do IDE a lista dos próximos eventos.

Nota

Para criar seu login na Embarcadero Community é muito simples: deve-se acessar o endereço que consta na seção **Links** e na parte superior direita clicar no link "Register". Este link levará para um formulário de cadastro onde deverá informar entre outras coisas seus dados pessoais e login e senha para acesso. A conta pode ser criada como conta individual, de uma empresa ou de estudante. Feito isso basta usar o usuário e senha cadastrados dentro do IDE do Rad Studio para se logar na Embarcadero Community pela Community Toolbar.

Além da Community Toolbar outro recurso muito interessante foi adicionado ao Rad Studio ainda na versão XE8 e que foi pouco divulgado, trata-se de um novo mecanismo que permite a instalação de componentes de uma forma extremamente mais simples e diretamente do IDE: o Getit Package Manager.

A partir de agora qualquer um que tenha um componente desenvolvido para o Delphi ou C++ poderá disponibilizá-lo neste que tende a ser o novo repositório de componentes, basta seguir

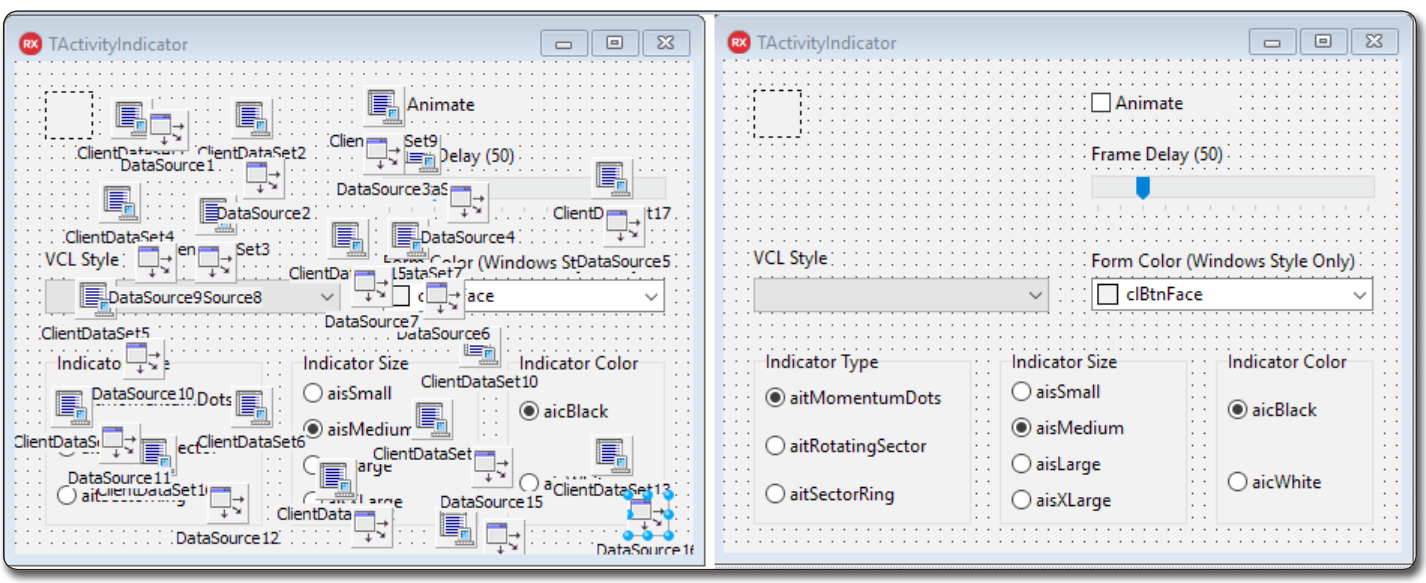


Figura 3. Hide Components

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

o protocolo que foi publicado na Embarcadero Community e cumprir todos os requisitos e seu componente estará disponível a todos os usuários, seja seu componente pago ou free. A **Figura 5** mostra a interface principal do Getit Package Manager.

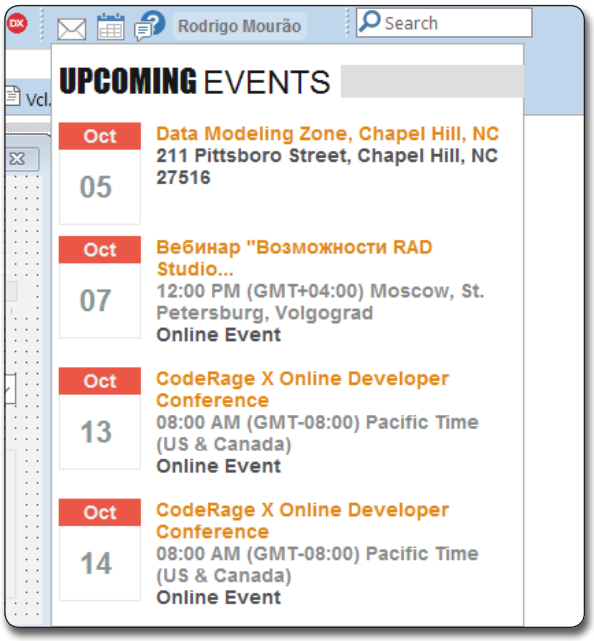


Figura 4. Community Toolbar

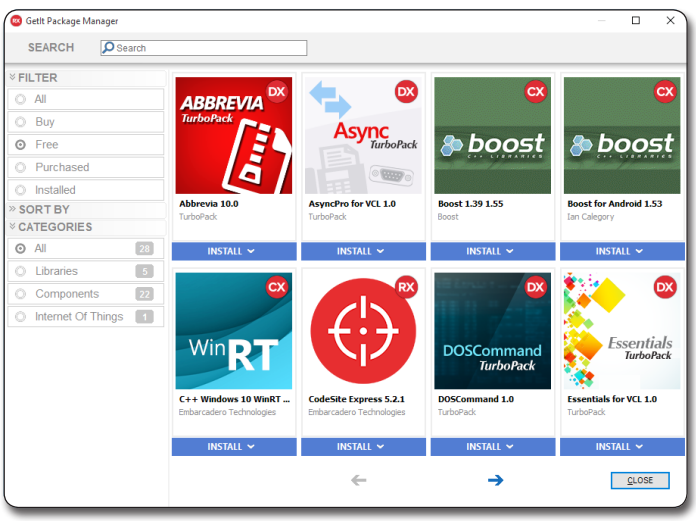
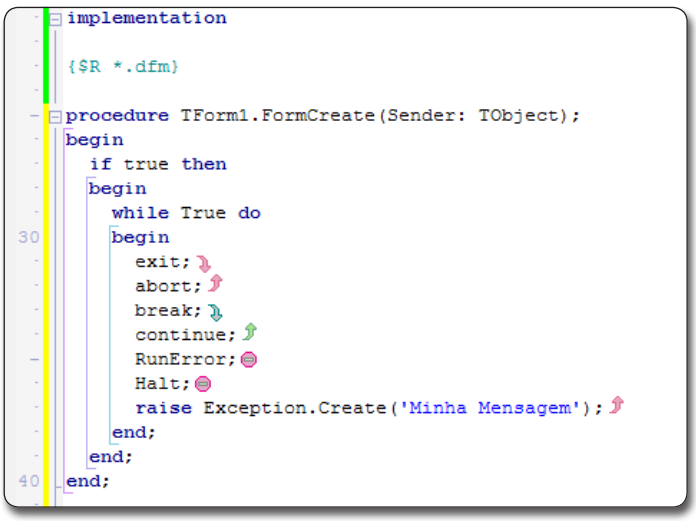


Figura 5. Getit Package Manager

Ainda falando sobre as novidades do IDE, temos a integração do plug-in Castalia, um velho conhecido dos desenvolvedores Delphi e que agora é parte integrante do IDE. Este plug-in adiciona algumas funcionalidades muito interessantes, principalmente no que diz respeito à produtividade. Quando era instalando como um plug-in, ele adicionava um menu no IDE e nele colocava todas as funcionalidades disponíveis para que pudéssemos ter acesso. Agora, como parte integrante do IDE, estas funcionalidades ficam distribuídas em outros locais.

Uma das funções interessantes deste plug-in é o controle visual de fluxo que ele adiciona ao nosso código, isso facilita e muito a leitura do código e nos dá uma noção de como o mesmo irá se comportar em tempo de execução. Veja na **Figura 6** alguns exemplos deste controle em pleno funcionamento, note como as linhas verticais delimitando cada bloco "Begin-End" facilitam a leitura do código, assim sabemos exatamente qual "End " corresponde a qual "Begin".



```
implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  if true then
  begin
    while True do
    begin
      exit;
      abort;
      break;
      continue;
      RunError;
      Halt;
      raise Exception.Create('Minha Mensagem');
    end;
  end;
end;
```

Figura 6. Controle Visual do Fluxo do Código

Estas são apenas algumas das muitas novidades do IDE que foram incorporadas nesta nova versão ou na versão anterior, mas podem não ter sido bem divulgadas, e é importante citar que a cada nova versão que é lançada novos recursos são adicionados.

Novidades de produtividade

Além das novidades do IDE, esta nova versão trouxe também novidades relacionadas à produtividade de trabalho em equipe. Vamos começar com uma ferramenta muito interessante que estava disponível no plug-in Castalia e agora faz parte do Delphi, o Sync Prototypes. Com este recurso podemos sincronizar automaticamente a implementação de suas rotinas que estão na seção "implematation" do seu código com sua declaração na seção "interface". Note que não estamos tratando do já conhecido Ctrl+Shift+C (Complete Class at Cursor) que ao ser invocado faz a implementação do código automaticamente. Esta nova funcionalidade tem como o objetivo de sincronizar seu código adicionando ou removendo parâmetros na declaração quando a alteração é feita na implementação ou vice-versa.

Além de modificações nos parâmetros, podemos alterar o tipo de um método de procedure para function e o Sync Prototypes se encarrega de mudar onde for necessário para que seu código compile sem erros. Para utilizar o recurso é muito simples, basta que após efetuar a modificação desejada, seja na declaração ou implementação, pressione Shift + Ctrl + Alt + P e o Sync Prototypes efetuará as alterações necessárias. Veja na **Figura 7** o método alterado na

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

seção “interface” do código e após invocar o Sync Prototypes o método alterado na seção implementation do código.

Além do Sync Prototypes, outro recurso foi herdado do plug-in Castalia, o “Project Statistics”. Este recurso coleta algumas informações ao longo do desenvolvimento do projeto e as armazena para montar um gráfico com os dados de tempo gasto em atividades especificas no projeto. Com ele sabemos exatamente quanto tempo gastou em um projeto, o que pode lhe ajudar a cobrar, por exemplo, por um serviço de manutenção ou ao menos saber se um orçamento dado por você de fato está de acordo com o que está sendo realizado.

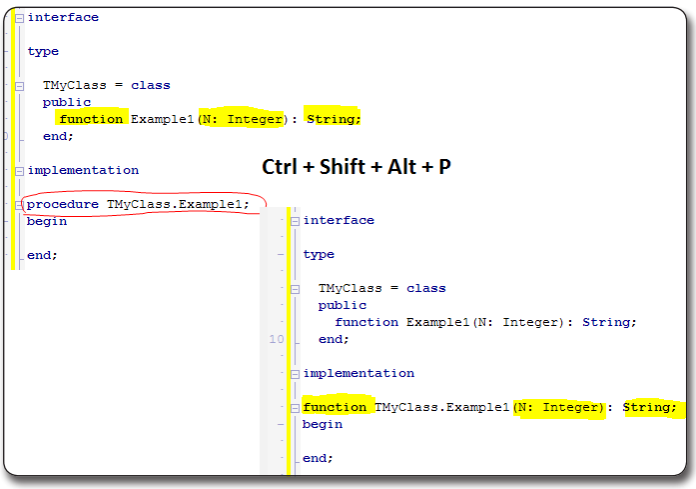


Figura 7. Sync Prototypes

O mais legal deste recurso é que ele separa este tempo gasto em atividades específicas, como tempo editando código, tempo gasto trabalhando no visual da aplicação, tempo compilando e até tempo depurando a aplicação. Veja na **Figura 8** um exemplo de estatística de um projeto, observe que do tempo total de 47 minutos gastos no projeto, 14 minutos foram codificando, seis trabalhando no visual da aplicação, quatro minutos inspecionando o código, dois minutos compilando o projeto e outros 19 minutos em outras atividades, que neste caso pode ser ociosidade, por exemplo.

Outro recurso muito legal e que de fato aumenta a produtividade é o Multi-Paste. Quando temos um componente de acesso a dados que executa comandos SQL em tempo de execução, não são raras as vezes em que temos que alterar o conteúdo deste comando SQL no código. Geralmente nós escrevemos o comando SQL em alguma ferramenta SQL qualquer e depois copiamos e colocamos esta instrução SQL para nosso código.

A parte chata está justamente neste último passo, isso porque na maioria das vezes temos que colocar este comando SQL, que não possui apenas uma única linha, dentro da propriedade “CommandText” ou “SQL” de algum Data Set e aí vem aquela perda de tempo formatando o código para que o comando SQL fique corretamente dentro dos parênteses e aspas simples exigidos pela sintaxe da linguagem. Com o recuso de Multi-Paste podemos colar um comando SQL com várias linhas em seu código já

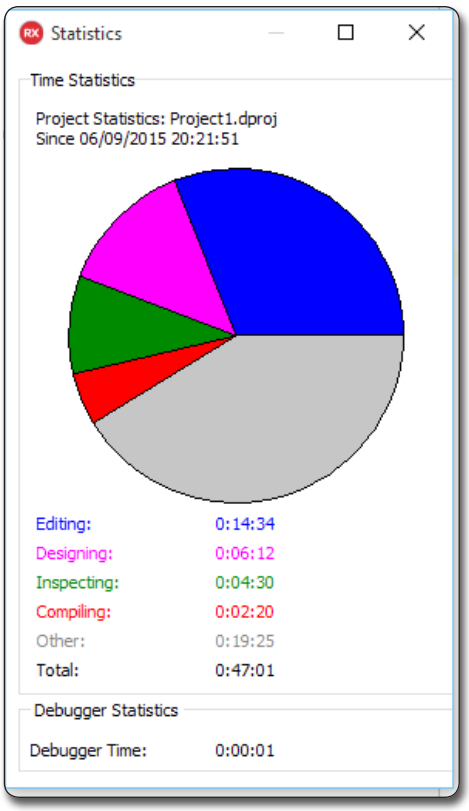


Figura 8. Project Statistics

formatando para o padrão exigido pelo componente Data Set que se está utilizando.

Veja na **Figura 9** um exemplo onde estamos colando um comando select com cerca de 30 linhas num bloco de código que utiliza um componente FDQuery para executar tal comando. Note que na tela do Multi-Paste foi solicitado que fosse incluída antes de cada linha do comando SQL a expressão “FDQuery1.Sql.Add(‘” e após cada linha a expressão “);”. O resultado é excelente, o comando SQL é incluído no código já formatado de acordo com o que o componente FDQuery exige.

Outro recurso que auxilia no dia a dia do desenvolvedor e com isso aumenta a produtividade é a nova barra de ferramentas, a “Navigation Toolbar”. Esta é uma barra de ferramentas que permite a navegação rápida não só entre as seções de uma Unit, ou seja, navegar entra a seção de declaração e implementação, mas sim em todo seu conteúdo. Ela consegue mapear os tipos declarados em uma Unit e quando estes tipos são classes ele também consegue mapear os métodos contidos dentro destas classes. Isso facilita o nosso trabalho principalmente quando lidamos com classes em nossos projetos que possuem muitos métodos e temos que navegar entre eles.

As **Figuras 10** e **11** mostram o uso da Navigation Toolbar dentro da Unit Vcl.Controls, que possui 17703 linhas. Veja na **Figura 10** que a Navigation Toolbar listou todos os tipos declarados na Unit, sejam classes, constantes, ponteiros ou records. Ao selecionar um tipo qualquer disponível nesta Unit, o cursor do mouse é direcionamento para o local exato onde o tipo selecionado está declarado.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

No combo ao lado deste que usamos para navegar pelos tipos é possível navegar pelos métodos contidos na Unit, sejam métodos públicos ou métodos declarados dentro de uma das classes existentes. A **Figura 11** mostra os métodos públicos da Unit sendo listados.

Vale ressaltar que a barra de navegação é uma ferramenta de mão dupla, ou seja, se selecionado uma classe ou método na barra de navegação, pode-se navegar até a declaração. Da mesma forma, à medida que se percorre o código os tipos e métodos são alterados automaticamente na barra de navegação para refletir o local exato onde se está posicionado.

Para fechar o assunto de produtividade, e mesmo sabendo que estes são apenas alguns dos recursos, vamos falar sobre o elemento Multi Device Preview. Todos sabemos que o Delphi é agora uma ferramenta que nos permite desenvolver para múltiplas plataformas, e entre estão o iOS e Android, permitindo que a aplicação seja executada em uma variedade de dispositivos.

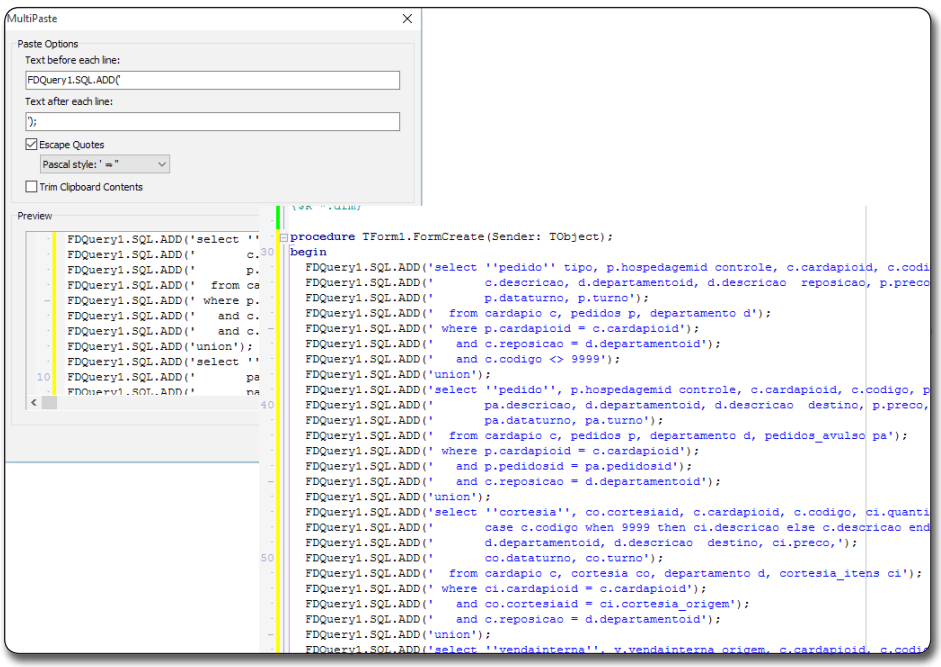


Figura 9. MultiPaste

No caso do iOS essa questão está um pouco mais sob controle, isto é, como o iOS roda apenas em iPhone, sabemos exatamente quantos e quais são os dispositivos no mercado que rodam o sistema da Apple.

Já para os dispositivos Android, essa variação é bem grande. É quase impossível saber com exatidão quantos e quais são os modelos de smartphones que hoje rodam o sistema Android. E o que interessa na hora do desenvolvimento é saber sobre estas variações.

Se vamos desenvolver para dispositivos móveis, temos que garantir que a identidade visual da nossa aplicação se manterá inalterada independente da resolução da tela dos dispositivos e essa não é uma questão secundária. Como ficaria caro manter um dispositivo de cada para testes, o Delphi trouxe um recurso muito interessante que nos permite ter uma visão em escala real da nossa aplicação em vários formatos de tela, e mais, se um formato desejado não estiver disponível nós podemos acrescentar sem problema algum.

A **Figura 12** ilustra o painel Multi Device Preview em ação, mostrando como se comportará uma aplicação mobile em cinco dispositivos diferentes.

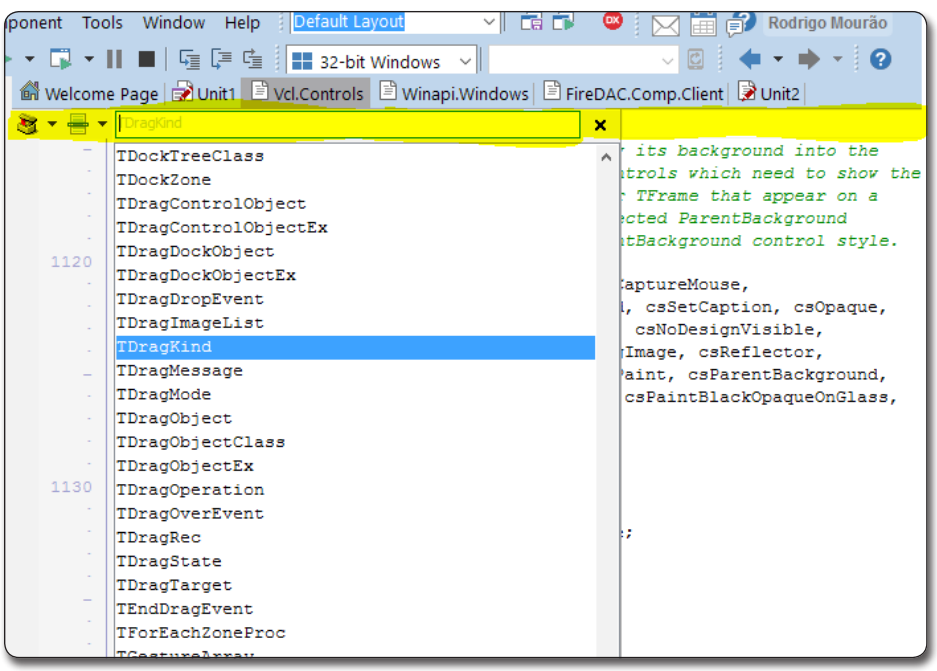


Figura 10. Navigation Toolbar

Novidades para o Windows 10

As novidades não se restringem apenas a novos componentes com suporte a Windows 10, mas também a novos estilos padrão do sistema operacional, novas funções na RTL, novas units para acesso à API do Windows, incluindo acesso à nova API WinRT. Vamos então começar falando sobre os novos estilos.

Já há algumas versões atrás foi adicionado ao Delphi o recurso de estilo visuais para as aplicações, mais em função do novo framework FireMonkey. Por exemplo, não é possível trocar a cor de um form pela propriedade color, na verdade quando

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

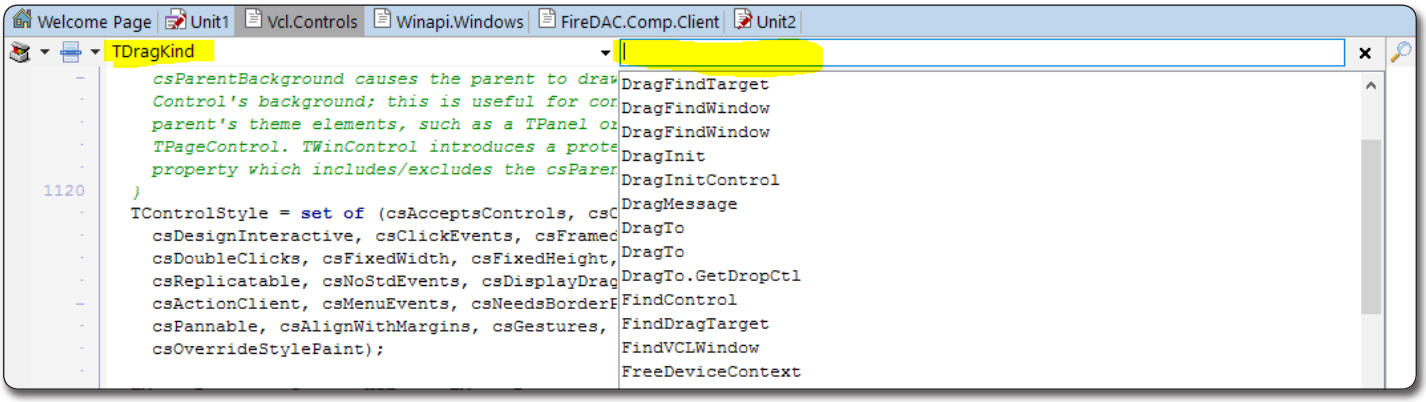


Figura 11. Listando métodos públicos

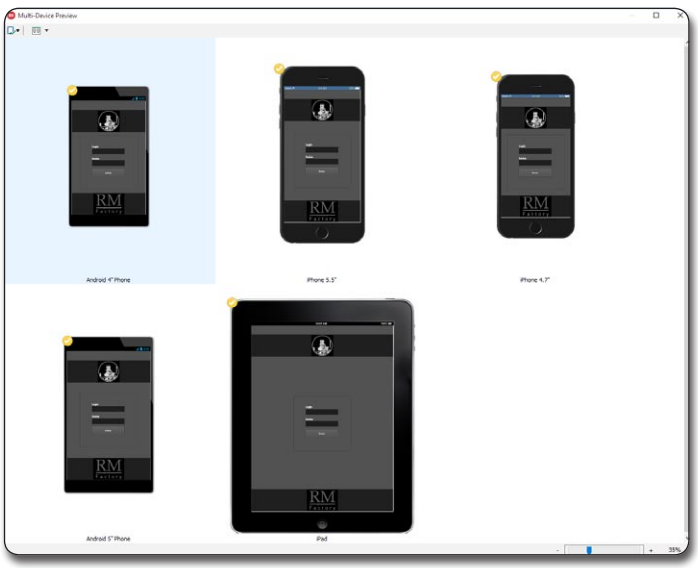


Figura 12. Multi Device Preview

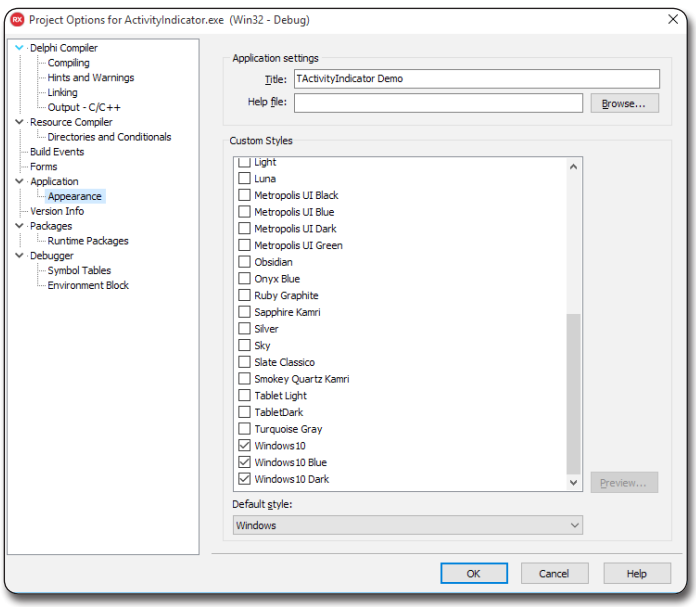


Figura 13. Lista de estilos disponíveis

estamos falando de aplicação FireMonkey a propriedade color nem sequer existe no formulário (**BOX 2**), todo o visual de uma aplicação FireMonkey deve ser gerido por estilos. Não é o foco aqui explicar como criar novos estilos, mas sim como acessar os novos estilos visuais com look and feel do Windows 10.

A primeira coisa que se deve fazer é, depois de criar um projeto, acessar o Menu Projects->Option e nas opções do projeto acessar o item Appearance dentro de Application. Nesta opção temos a lista de todos os estilos atualmente suportados pelo Delphi. A **Figura 13** mostra a lista de estilos.

BOX 2. Design de Aplicações no FireMonkey

Embora não haja mais a propriedade color no formulário do FireMonkey, ainda assim é possível alterar sua cor. Basta acessar a propriedade Fill do formulário na subpropriedade Color e selecionar a cor desejada. Porém é fato que embora mais simples e objetivo, este não deve ser o caminho a ser seguido. Com os recursos de estilo conseguimos manter a identidade visual de suas aplicações. Pode não parecer, mas uma variação de tom em uma das cores de um formulário para o outro ou de um botão para o outro é motivo de reprovação por parte dos clientes, mesmo que inconscientemente. Assim sempre que possível use os styles personalizados.

Os últimos estilos adicionados são justamente os estilos para Windows 10, Windows 10 Blue e Windows 10 Dark. Ao marcar estas opções habilitamos o projeto a utilizar estes estilos e mudar completamente seu Visual. Vamos fazer um teste. Crie um projeto VCL novo e adicione no form alguns controles como edit, buttons, labels, etc. Coloque ao menos um botão. Vá na opção que acabamos de ver e habilite os três estilos do Windows 10. Feito isso, vá no evento OnClick de um dos botões e adicione o código da **Listagem 1**.

Nota

Também foi adicionado a essa nova versão o estilo OnyxBlue, como vemos na **Figura 13**.

Note que para configurar o estilo em tempo de execução utilizamos a classe TStyleManager, que para ser acessada requer a inclusão da unit Vcl.Themes na cláusula uses. Feito isso, basta invocar o método SetStyle e passar o nome do estilo que se quer carregar. Na **Listagem 1** comentamos o carregamento dos demais estilos e deixamos ativo apenas o estilo Windows Blue, como

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

teste troque a linha descomentada por outra e veja o resultado. A **Figura 14** mostra uma aplicação com estilo padrão do Windows e a mesma aplicação com estilo Windows10 Blue.

Listagem 1. Carregando estilos dinamicamente

```
01 procedure TFrmPrincipal.Button1Click(Sender: TObject);
02 begin
03  //TStyleManager.SetStyle('Windows');
04  //TStyleManager.SetStyle('Windows10');
05  TStyleManager.SetStyle('Windows10 Blue');
06  //TStyleManager.SetStyle('Windows10 Dark');
07 end;
```

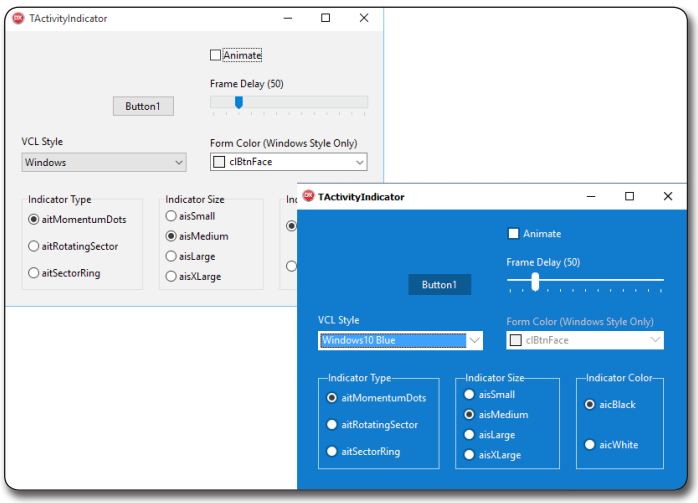


Figura 14. Estilos do Windows 10

Além dos estilos do Windows 10 temos também alguns novos controles exclusivos para uso nesse sistema operacional, são controles que utilizam novos recursos da API para serem renderizados na tela. Se buscar na paleta de componentes, veremos que há uma nova paleta chamada "Windows 10" e nesta paleta temos os seguintes novos componentes:

- TActivityIndicator: utilizado para indicar que uma aplicação está em processamento, exibindo uma animação no estilo do Windows 10;
- TToggleSwitch: funciona semelhante a um checkbox com estados On e Off, porém com inferface otimizada para o toque e no estilo do novo sistema operacional;
- TSearchBox: caixa de texto que integra um botão e é utilizada para implementar ações de busca dentro do sistema. Após digitado o texto, pode-se clicar no botão ou pressionar Enter e isso irá disparar evento no qual deve ser feita a busca;
- TRelativePanel: painel para agrupamento de componentes visuais que permite organizá-los na tela de forma relativa um ao outro;
- TSplitView: menu lateral que pode ser exibido e ocultado dinamicamente.

A **Figura 15** mostra a nova paleta de componentes Windows 10.

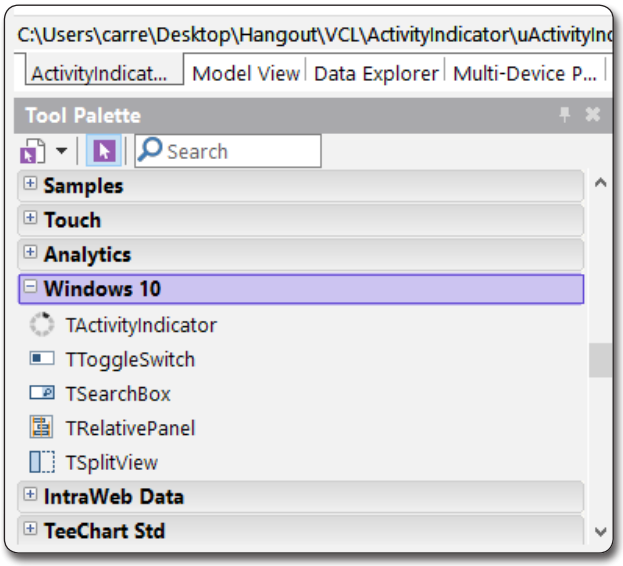


Figura 15. Paleta Windows 10

Neste artigo não será possível falar de todos os componentes desta nova paleta, então vamos conhecer um dos mais interessantes, tanto do ponto de vista de funcionalidade, quanto do ponto de vista visual. O componente é o TSplitView, ele nos permite criar um menu retrátil no estilo Windows 10, bem parecido com o que já ocorre com as últimas versões do office.

Para vermos na prática seu funcionamento, vamos criar um novo projeto VCL e no form principal da aplicação adicionar os seguintes componentes:

- Um TSplitView;
- Três TButtons;
- Um TCategoryButtons (dentro do Split View);
- Um TImageList.

Adicione dentro TImageList três imagens que irão representar nossos três menus. Aponte a propriedade Images do TCategoryButtons para esta ImageList. Em seguida acesse o TCategoryButtons e crie uma nova categoria através da propriedade "Categories". Nesta nova categoria crie três novos itens, por exemplo, os captions "Início", "Cadastro" e "Relatórios". Escolha uma imagem para cada um destes itens através da propriedade ImageIndex de cada um dos itens da categoria que criamos.

Para darmos um ar de Windows 10 ao nosso exemplo, altere a cor do componente TSplitView para $00FF8000. Antes de vermos como deve estar nosso exemplo, altere o caption de cada um dos três botões para "Compact", "Collapse" e "Open". Agora sim, nosso exemplo deve estar com o visual parecido com o da **Figura 16**.

Vamos então utilizar algumas funcionalidades interessantes deste componente. Para começar é importante citar que este componente possui dois métodos principais, um para abrir o componente e outro para fechá-lo. Estes métodos são "Open" e "Close" respectivamente, e com eles pode-se controlar como o componente é fechado e aberto informando se haverá ou não animação e qual será a velocidade desta animação, caso opte por animar o evento.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

CURSOS ONLINE

Clube Delphi

A Revista Clube Delphi oferece aos seus assinantes
uma série de Cursos Online de alto padrão de qualidade.

CONHEÇA ALGUNS DOS CURSOS

- Curso de Multicamadas com Delphi e DataSnap
- Delphi para Iniciantes
- Criando componente Boleto em Delphi
- Loja Virtual em Delphi Prism

Para mais informações :

www.devmedia.com.br/cursos/delphi

(21) 3382-5038

http://www.devmedia.com.br

DEVMEDIA

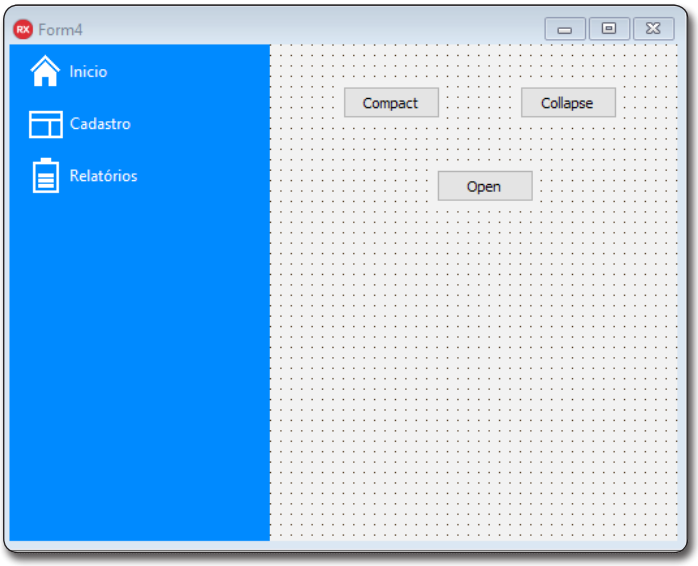


Figura 16. Exemplo de uso do Split View

Vamos fazer um exemplo bem simples. Vamos determinar a forma como o componente será fechado e então fechá-lo. Para isso, programe o evento OnClick do botão "Compact" e implemente o código da **Listagem 2**. Na prática apenas fechamos o componente, porém antes disso informamos de que forma queremos que o componente seja fechado, neste caso de forma compacta.

O código do botão "Collapse" terá o mesmo objetivo, fechar o componente, porém desta vez ele será fechado de forma completa e não compacta. O código da **Listagem 3** mostra a implementação do evento OnClick do botão "Collapse"

Listagem 2. Código do botão Compact

```
01 procedure TForm4.BtnCompact(Sender: TObject);
02 begin
03   SplitView1.CloseStyle := TSplitViewCloseStyle.svcCompact;
04   SplitView1.Close;
05 end;
```

Listagem 3. Código do Botão Collapse

```
01 procedure TForm4.BtnCollapse(Sender: TObject);
02 begin
03   SplitView1.CloseStyle := TSplitViewCloseStyle.svcCollapse;
04   SplitView1.Close;
05 end;
```

Agora falta a implementação do botão "Open". Para isso, implemente no evento OnClick deste botão de acordo com o código da **Listagem 4**. Feito isso, nosso exemplo está finalizado e pronto para ser executado. Cabe ressaltar que é um exemplo extremamente simples que serve apenas para mostrar de forma prática as possibilidades que este componente nos oferece. Execute a aplicação e clique no botão "Compact". Veja **na Figura 17** como o nosso menu se apresenta de forma compacta ao ser fechado.

Clique no botão "Open" para que o menu volte à forma normal e após isso clique no botão "Collapse" para que o menu seja fechado de forma completa. Veja na **Figura 18** o comportamento do menu fechado de forma completa.

Na pasta de exemplos que são disponibilizados durante a instalação do Rad Studio existe um exemplo mais completo mostrando outras possibilidades para o uso do Split View. Vale a pena dar uma olhada.

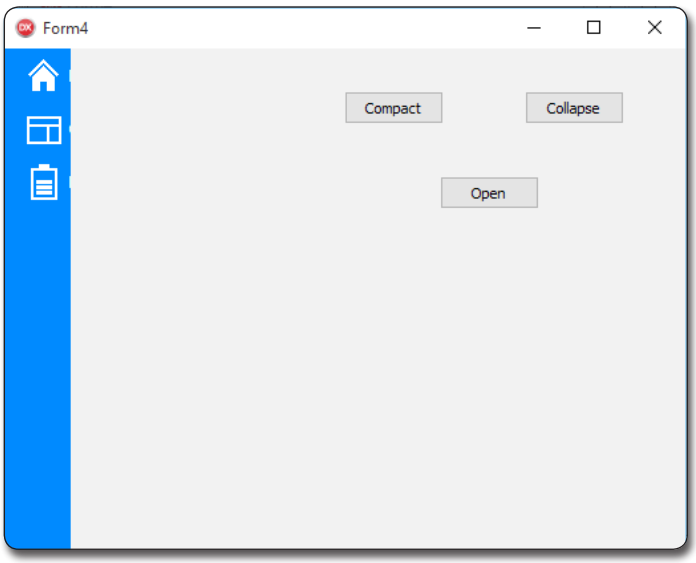


Figura 17. Split View Compact

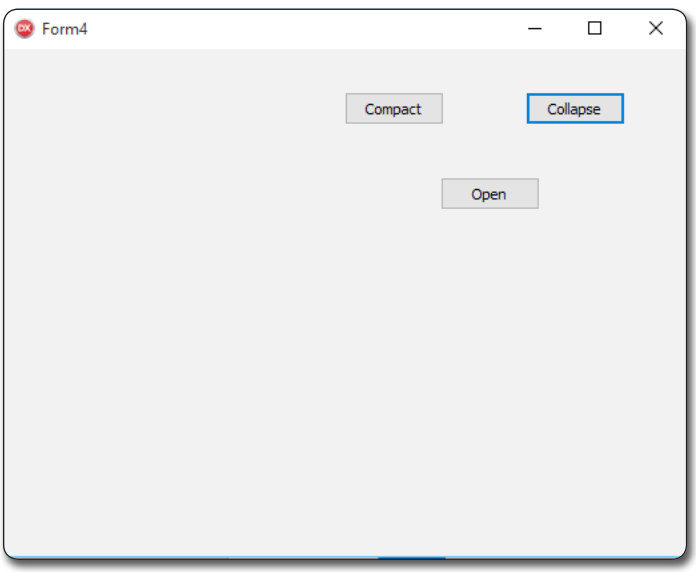


Figura 18. Split View Collapse

Listagem 4. Código do botão Open

```
01 procedure TForm4.BtnOpen(Sender: TObject);
02 begin
03   SplitView1.Open;
04 end;
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Sharing Contract

O Windows, desde a versão 8, trabalha com o conceito de Contratos, que são formas de a aplicação se comunicar com o sistema operacional e com outras aplicações para executar tarefas comuns, como compartilhamento de dados. É comum encontrar nas aplicações modernas do Windows uma opção "Compartilhar", que quando acionada exibe uma lista de aplicações com as quais a ação pode ser concluída. Por exemplo, para compartilhar uma foto pode-se utilizar aplicações de troca de mensagens instantâneas ou e-mail.

O componente TSharingContract trouxe às aplicações desenvolvidas com o Delphi 10 Seattle a possibilidade de implementar esse contrato, compartilhando dados com outras aplicações instaladas no sistema operacional, desde que sejam aplicações no estilo moderno (Windows Store).

Para testar esse componente, crie um novo projeto do tipo VCL Forms Application e adicione um componente do tipo TSharingContract e um botão. No código do evento OnClick do botão adicione o código visto na **Listagem 5**.

Com isso, ao clicar no botão devemos ter um resultado semelhante ao que mostra a **Figura 19,** com as aplicações disponíveis para compartilhamento sendo listadas na Charm Bar do Windows.

Listagem 5. Compartilhando dados com TSharingContract

```
01 procedure TForm1.Button1Click(Sender: TObject);
02 begin
03   SharingContract1.DataTitle := 'Compartilhando dados';
04   SharingContract1.HTML := '<h1>Delphi 10 Seattle</h1>' +
05                            '<h2>Conheça as novidades da versão</h2>' +
06                            '<p>Aqui vemos as novas funcionalidades do Delphi 10.</p>';
07   SharingContract1.InitSharing;
08 end;
```

Observe que no topo da barra de compartilhamento já aparece o título que definimos no código "Compartilhando dados", isso ajuda o usuário a identificar qual aplicação está invocando essa ação e qual seu objetivo. Clicando sobre uma delas, cada aplicação reagirá de uma forma distinta. Por exemplo, a aplicação de Email utiliza o título como assunto de uma nova mensagem e o conteúdo como corpo desta, permitindo enviar rapidamente um novo e-mail, como mostra a **Figura 20.**

São diversas as possibilidades de aproveitamento dessa funcionalidade, por exemplo, ao filtrar dados em uma aplicação financeira, o usuário poderia facilmente compartilhar os resultados por e-mail utilizando a aplicação Email que já estaria configurada no seu sistema operacional. Além de conteúdo HTML, como o que foi usado no exemplo, também é possível compartilhar texto no formato RTF, links e imagens. Quais aplicações irão responder à ação de compartilhamento dependerá de como elas implementam o contrato de busca, pois dependendo do conteúdo que está sendo compartilhado uma aplicação pode ou não ser compatível com o contrato.

É muito importante conhecer estes novos componentes para criação de interface estilo Windows 10. Isso porque sempre que a Microsoft lança uma nova interface, o novo padrão rapidamente é adotado pelos sistemas, isso para que as aplicações sejam as mais amigáveis possíveis com as telas do próprio sistema operacional. Por exemplo, por que não criar nossas telas de cadastro utilizando agora o TSplitView e os nossos usuários terem a partir de agora uma nova experiência de usabilidade?

Em cada novo lançamento sempre temos muitas melhorias no framework FireMonkey, melhorias e novidades na tecnologia de acesso a dados Firedac, livebindings e recentemente recursos para desenvolvimento para internet das coisas (IoT).

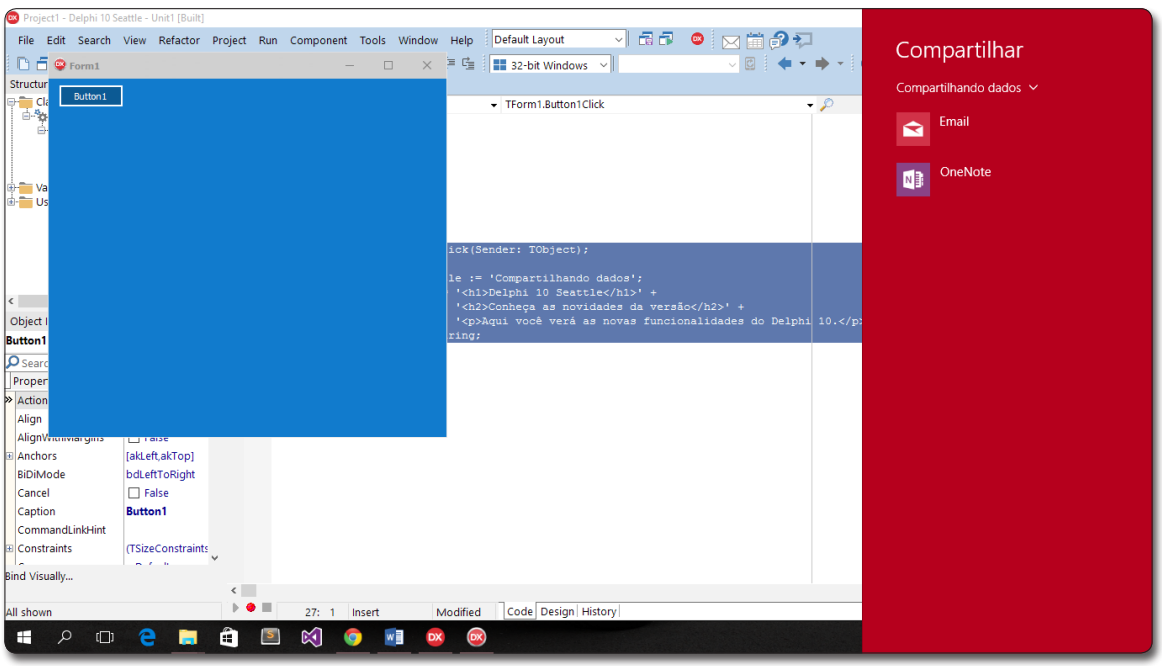


Figura 19. Aplicações disponíveis para compartilhar dados

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

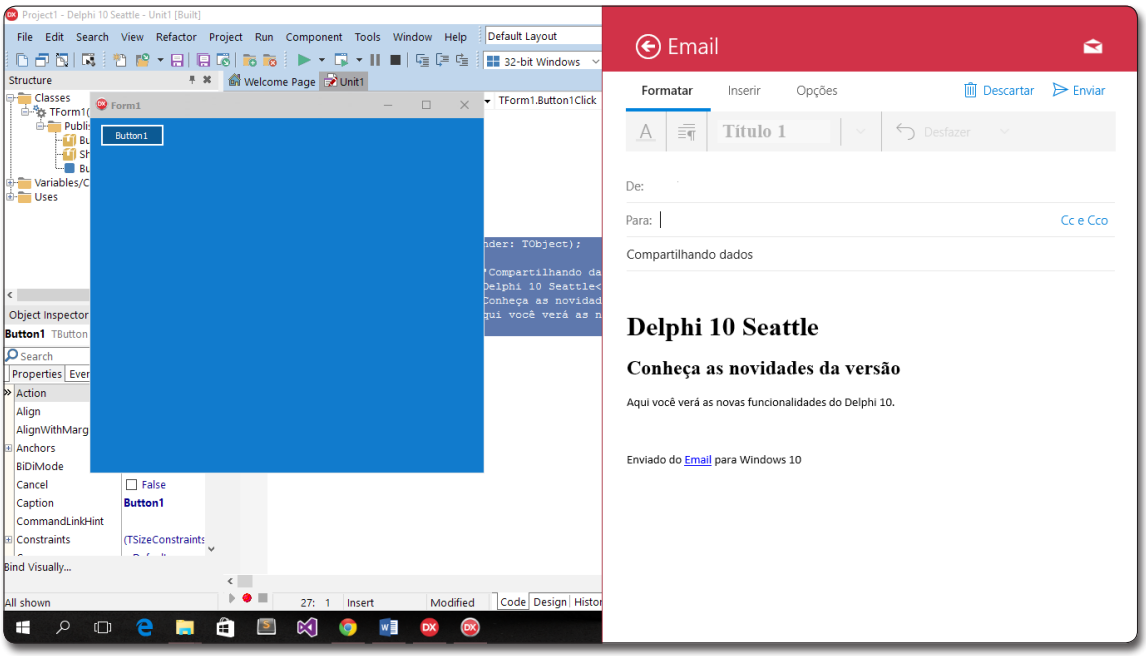


Figura 20. Utilizando aplicação Email para compartilhar dados

No FireMonkey, entre as novidades podemos destacar a possibilidade que a FMX possui de criar aplicações nativas para Windows 10. Além disso, agora é possível implementar recursos de arrastar e soltar (Drag and Drop) em aplicações para OSX. Os mesmos estilos Windows 10 vistos para VCL também estão disponíveis para FireMonkey. Agora os componentes FMX também suportam hints em aplicações desktop, o que não era possível até o XE8. Uma série de controles visuais agora são reenderizados de forma nativa em várias plataformas.

O mesmo ocorre para o FireDac, que é a tecnologia padrão de acesso a dados do Rad Studio. No FireDac podemos destacar melhorias de otimização nos componentes ETL, que são componentes utilizados para migração de grandes massas de dados. Outra novidade que está muito interessante é o suporte do FireDac ao MongoDB, o banco de dados NoSQL mais famoso atualmente. Agora podemos desenvolver aplicações que acessam diretamente este banco dados, pois o Rad Studio incluir um drive para conexão direta ao MongoDB. Há também muitas novidades da RTL, compilador, depurador, entre outros.

Além das novas funcionalidades, também foram feitas melhorias em recursos já existentes, como nos componentes para acesso a serviços na nuvem, que foram atualizados para suportar as mais recentes versões das APIs do Amazon Web Services e o Microsoft Azure. O trabalho com JSON foi otimizado, por exemplo, com o suporte ao formato BSON que é utilizado pelo MongoDB. A unit System.Hash agora conta com a classe THashSHA2, que implementa a família de algoritmos de criptografia do tipo SHA-2. Existem outras várias alterações menores, porém bastante significativas, dependendo do contexto de uso, como para projetos Android e iOS.

Autor

Rodrigo Carreiro Mourão
rodrigomourao@rodrigomourao.com.br
Graduado em gestão de TI com especialização em gerência de projetos (PMI), Governança de TI (COBIT) e Gestão de Serviços de TI (ITIL) pelo instituto Infnet. Grande entusiasta de metodologias ágeis, padrões de projetos e gestão por processos. Ultimamente tem se dedicado a desbravar a área de Software Quality Assurance para ajudar a fomentar esta boa prática em empresas de todos os tamanhos e segmentos.

Links:

Wiki Embarcadero
http://docwiki.embarcadero.com

Política de licenciamento para atualizações do RAD Studio
http://www.embarcadero.com/products/rad-studio/update-subscription

20 anos de Delphi
http://www.embarcaderobr.com.br/Delphi/#historia

Embarcadero Community
http://community.embarcadero.com/

Submetendo componentes para o repositório Getit
http://community.embarcadero.com/article/news/16027-submitting-libraries-to-getit

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

CURSOS ONLINE

Feita para Desenvolvedores de Software e DBAs
SQL magazine

A Revista SQL Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

CONHEÇA OS CURSOS MAIS RECENTES:

- **Cursos: Curso de noSQL (Redis) com Java**
- **Desenvolvimento para SQL Server com .NET**
- **Curso PostgreSQL - Treinamento de banco de dados (Curso Básico)**

Para mais informações :

http://www.devmedia.com.br/cursos/banco-de-dados

http://www.devmedia.com.br

DEVMEDIA

Explorando APIs do Windows em Delphi – Parte 2

Este artigo faz parte de uma série

Nesta segunda parte do artigo iremos explorar mais algumas categorias de recursos da API do Windows que em algum momento a maior parte dos desenvolvedores precisa utilizar, como:

- Mouse;
- Impressoras;
- Desenhando Formas;
- Manipulando Sons;
- Usando ShellExecute;
- Manipulando a Ajuda do Windows.

Mouse

Nesta categoria temos funções que nos permitem obter informações sobre o cursor do mouse, bem como definilas, simulando movimentos e cliques dos botões através de parâmetros que indicam o tipo de ação executada, as coordenadas e quais botões foram utilizados. Entre essas funções, temos algumas que merecem maior destaque por serem mais utilizadas.

Função mouse_event

A função mouse_event está declarada na DLL "user32 .dll" e é utilizada para sintetizar o movimento do mouse e o clique de seus botões, enviando essas informações para a entrada do sistema, de forma a simular uma ação real sobre o mouse. Sua assinatura é a que vemos na **Listagem 1**.

Os argumentos dessa função são descritos a seguir.

• **dwFlags**: Esse parâmetro representa uma combinação de valores (flags) que representam as informações de movimento e clique do mouse que devem ser enviados para o sistema, simulando uma ação real. Essas flags são representadas pelas seguintes constantes:

Fique por Dentro

Neste artigo continuaremos explorando os recursos da API do Windows para executar algumas tarefas comuns no dia a dia do programador, como a manipulação de impressoras, mouse e sons diretamente. O conhecimento dessa API é bastante útil por permitir ao desenvolvedor agregar às suas aplicações algumas funcionalidades que estão disponíveis diretamente no sistema operacional, sem precisar reescrevê-las ou utilizar componentes de terceiros.

Listagem 1. Assinatura da função mouse_event

```
01 (ByVal dwFlags As Long,
02  ByVal dx As Long,
03  ByVal dy As Long,
04  ByVal dwData As Long,
05  ByVal dwExtraInfo As Long)
```

- **MOUSEEVENTF_ABSOLUTE**: Essa flag indica que os parâmetros DX e DY contêm as coordenadas absolutas do mouse. No sistema de coordenadas utilizado pela função, o canto da tela superior esquerdo da tela tem as coordenadas (0,0) e o canto inferior direito tem coordenadas (65535,65535), independentemente do tamanho da tela real. Se este sinalizador não for definido, DX e DY contêm coordenadas relativas, cuja quantidade de movimento real depende das configurações de velocidade e aceleração atuais do mouse;
- **MOUSEEVENTF_LEFTDOWN**: Indica que o botão esquerdo do mouse foi pressionado;
- **MOUSEEVENTF_LEFTUP**: Indica que o botão esquerdo foi liberado;
- **MOUSEEVENTF_MIDDLEDOWN**: O botão do meio foi pressionado;
- **MOUSEEVENTF_MIDDLEUP**: O botão do meio foi liberado.
- **MOUSEEVENTF_MOVE**: Indica que o mouse se moveu. Neste caso, os parâmetros DX e DY especificam a quantidade, ou a localização do movimento;

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

- **MOUSEEVENTF_RIGHTDOWN**: O botão direito foi pressionado;
- **MOUSEEVENTF_RIGHTUP:** O botão direito foi liberado;
- **MOUSEEVENTF_WHEEL**: Indica que a "roda" foi movida e o parâmetro dwData especifica a quantidade de movimento;
- **MOUSEEVENTF_XDOWN**: Um botão X foi pressionado. O parâmetro dwData identifica quais botões X;
- **MOUSEEVENTF_XUP**: Um botão X foi liberado. O parâmetro dwData identifica quais botões X.

• **dX**: Especifica a coordenada X absoluta do mouse ou a quantidade de movimento relativo ao eixo X. Para o movimento relativo, valores positivos movem para a direita e os valores negativos movem para a esquerda;

• **dY**: Especifica a coordenada Y absoluta do mouse ou a quantidade de movimento relativo ao eixo Y. Valores positivos movem para baixo e os valores negativos movem para cima;

• **dwData**: Se dwFlags contém MOUSEEVENTF_WHEEL, esse parâmetro especifica a quantidade de movimento da roda, em múltiplos inteiros de WHEEL_DATA. Valores positivos significam rotação para a frente e valores negativos significam rotação para trás. Se dwFlags contém qualquer MOUSEEVENTF_XDOWN ou MOUSEEVENTF_XUP, esse parâmetro indica qual botão X foi pressionado ou solto, podendo receber um dos seguintes valores:
- **Xbutton1**: O primeiro botão X foi pressionado ou solto;
- **Xbutton2**: O segundo botão X foi pressionado ou solto.

• **DwExtraInfo**: Um valor de 32 bits adicional associado com o evento mouse, que pode ser acessado através da função GetMessageExtraInfo quando for necessário.

É importante destacar que o argumento dwFlags pode receber um ou mais desses valores que foram apresentados, uma vez que representam flags. Quando for necessário passar mais de um valor, eles devem ser separados pelo operador OR, assim serão interpretados como um conjunto. Por exemplo, se utilizássemos a seguinte sintaxe, faríamos com que o cursor do mouse fosse movido para a posição (0,0):

```
mouse_event(MOUSEEVENTF_MOVE Or MOUSEEVENTF_ABSOLUTE, 0, 0, 0, 0);
```

Função GetDoubleClickTime

A função GetDoubleClickTime também está declarada em "user32.dll" e sua assinatura é a seguinte:

```
() As Long
```

Ou seja, é uma função que não recebe parâmetros e retorna um valor do tipo Long indicando o tempo máximo (em milissegundos) permitido entre cliques sucessivos do mouse para que o Windows interprete como um duplo clique.

Criando uma aplicação para manipular o mouse

Para praticar esses conceitos, vamos criar uma aplicação do tipo VCL Forms, cuja interface do form principal será com posta por cinco TLabels, cinco TButons, e um Tmemo, como vemos na **Figura 1.** Cada um desses botões irá realizar uma chamada à função mouse_event, passando em cada caso argumentos diferentes. Na **Listagem 2** temos o código do evento OnClick dos botões.

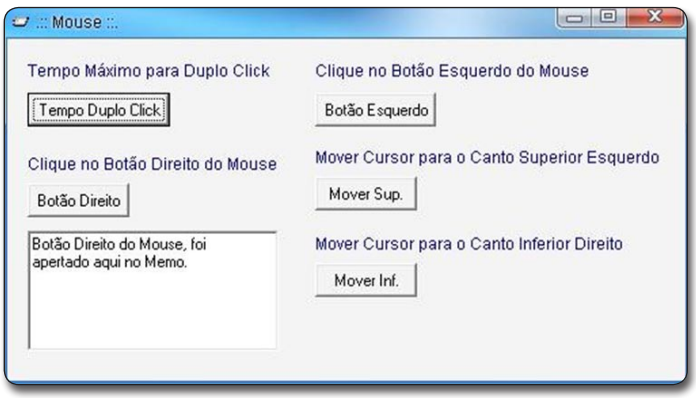


Figura 1. Tela do aplicativo para movimentação do mouse

Listagem 2. Implementando os códigos dos botões.

```
01 procedure TFrm_Principal.Btn_DpClickClick(Sender: TObject);
02 begin
03   ShowMessage('O Tempo Máximo em MileSegundos para duplo clique é '
04             + IntToStr(GetDoubleClickTime));
05 end;
06
07 procedure TFrm_Principal.Btn_ClickDirClick(Sender: TObject);
08 var
09   Pt: TPoint;
10 begin
11   Pt.x := 0;
12   Pt.y := 30;
13
14   mouse_event(MOUSEEVENTF_MOVE, Pt.x, Pt.y, 0, 0);
15   mouse_event(MOUSEEVENTF_RIGHTDOWN,Pt.x, Pt.y, 0, 0);
16   mouse_event(MOUSEEVENTF_RIGHTUP,Pt.x, Pt.y, 0, 0);
17 end;
18
19 procedure TFrm_Principal.Btn_ClickEsqClick(Sender: TObject);
20 var
21   Pt: TPoint;
22 begin
23   mouse_event(MOUSEEVENTF_MOVE Or MOUSEEVENTF_ABSOLUTE,
       400, 65000, 0, 0);
24
25   mouse_event(MOUSEEVENTF_LEFTDOWN, Pt.x, Pt.y, 0, 0);
26   mouse_event(MOUSEEVENTF_LEFTUP, Pt.x, Pt.y, 0, 0);
27 end;
28
29 procedure TFrm_Principal.Btn_MovSupClick(Sender: TObject);
30 begin
31   mouse_event(MOUSEEVENTF_MOVE Or MOUSEEVENTF_ABSOLUTE, 0, 0, 0, 0);
32 end;
33
34 procedure TFrm_Principal.Btn_MovInfClick(Sender: TObject);
35 begin
36   mouse_event(MOUSEEVENTF_MOVE Or MOUSEEVENTF_ABSOLUTE,
       65000, 65000, 0, 0);
37 end;
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Perceba que em alguns casos informamos mais de um valor para o parâmetro dwFlag, dessa forma obtemos o movimento desejado.

Impressora

Trabalhar com a impressora é uma tarefa comum em grande parte das aplicações comerciais desenvolvidas em Delphi. Na API do Windows temos uma função que facilita a interação com as impressoras que estão instaladas no sistema.

Função EnumPrinters

A função EnumPrinters está declarada em "winspool.drv" e a sua assinatura pode ser vista na **Listagem 3.**

Listagem 3. Assinatura da função EnumPrinters

```
01 EnumPrinters (
02   ByVal Flags As Long,
03   ByVal Name As String,
04   ByVal Level As Long,
05   ByVal pPrinterEnum As Long,
06   ByVal ByVal cdBuf As Long,
07   ByVal pcbNeeded As Long,
08   ByVal pcReturned As Long
09 ) As Long
```

Essa função encontra e retorna informações sobre uma ou mais impressoras a que o computador tem acesso, incluindo tanto as impressoras locais (fisicamente conectadas à máquina), quanto as impressoras de rede. Os argumentos dessa função são descritos a seguir.

• **Flags**: De forma semelhante à função mouse_event, o argumento Flags recebe um ou mais dos seguintes valores, que indicam que tipo de impressora se deseja listar:

- **PRINTER_ENUM_CONNECTIONS**: Obtém informações sobre as impressoras de rede com as quais o computador faz conexões;
- **PRINTER_ENUM_DEFAULT**: Obter informações sobre a impressora padrão do computador;
- **PRINTER_ENUM_LOCAL**: Obter informações sobre as impressoras locais (aqueles diretamente ligados ao sistema);
- **PRINTER_ENUM_NAME**: Lista informações sobre todas as impressoras sob o domínio de rede especificado pelo nome;
- **PRINTER_ENUM_NETWORK**: Obtém informações sobre todas as impressoras sob o domínio do computador na rede. Isso só funciona se a estrutura PRINTER_INFO_1 for passada no argumento Level;
- **PRINTER_ENUM_REMOTE**: Possui o mesmo funcionamento que PRINTER_ENUM_NETWORK;
- **PRINTER_ENUM_SHARED**: Obter informações sobre todas as impressoras com o atributo compartilhado.

• **Name**: Este argumento indica o nome de domínio de rede utilizado para pesquisar as impressoras, se for o caso, dependendo do valor passado no parâmetro Flags. Se esse parâmetro não for utilizado, deve-se passar uma string vazia;

• **Level**: Especifica qual estrutura deve ser usada para obter as informações das impressoras. Esse valor pode ser PRINTER_INFO_1, PRINTER_INFO_2 ou PRINTER_INFO_4;

• **pPrinterEnum**: Uma matriz que recebe toda a informação encontrada pela função. Após invocar a função, o valor desse argumento precisa ser copiado manualmente para uma estrutura PRINTER_INFO_ *;

• **cdBuf:** O tamanho em bytes da matriz passada como pPrinterEnum;

• **pcbNeeded:** Se a chamada à função for bem sucedida, esse argumento recebe o número de bytes de informação retornado pela função. Se a função não tiver êxito, esse argumento recebe o número de bytes que pPrinterEnum deve ter a fim de receber todas as informações;

• **pcReturned**: Recebe o número de impressoras encontradas pela função.

Uma situação comum em que se pode utilizar essa função é na listagem de impressoras para o usuário escolher em qual deve imprimir um relatório, por exemplo.

Criando uma aplicação para listar as impressoras

Criaremos agora uma aplicação VCL Forms cuja interface é composta apenas por um TListBox, e um TBitBtn, conforme vemos na **Figura 2.** Nesta aplicação listaremos as impressoras no listbox ao clicar no botão, de acordo com o código da **Listagem 4.**

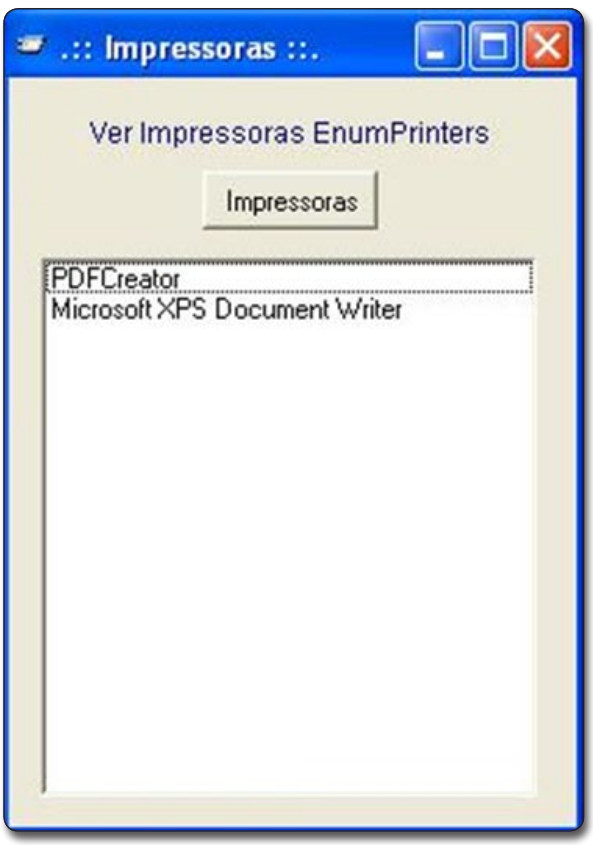
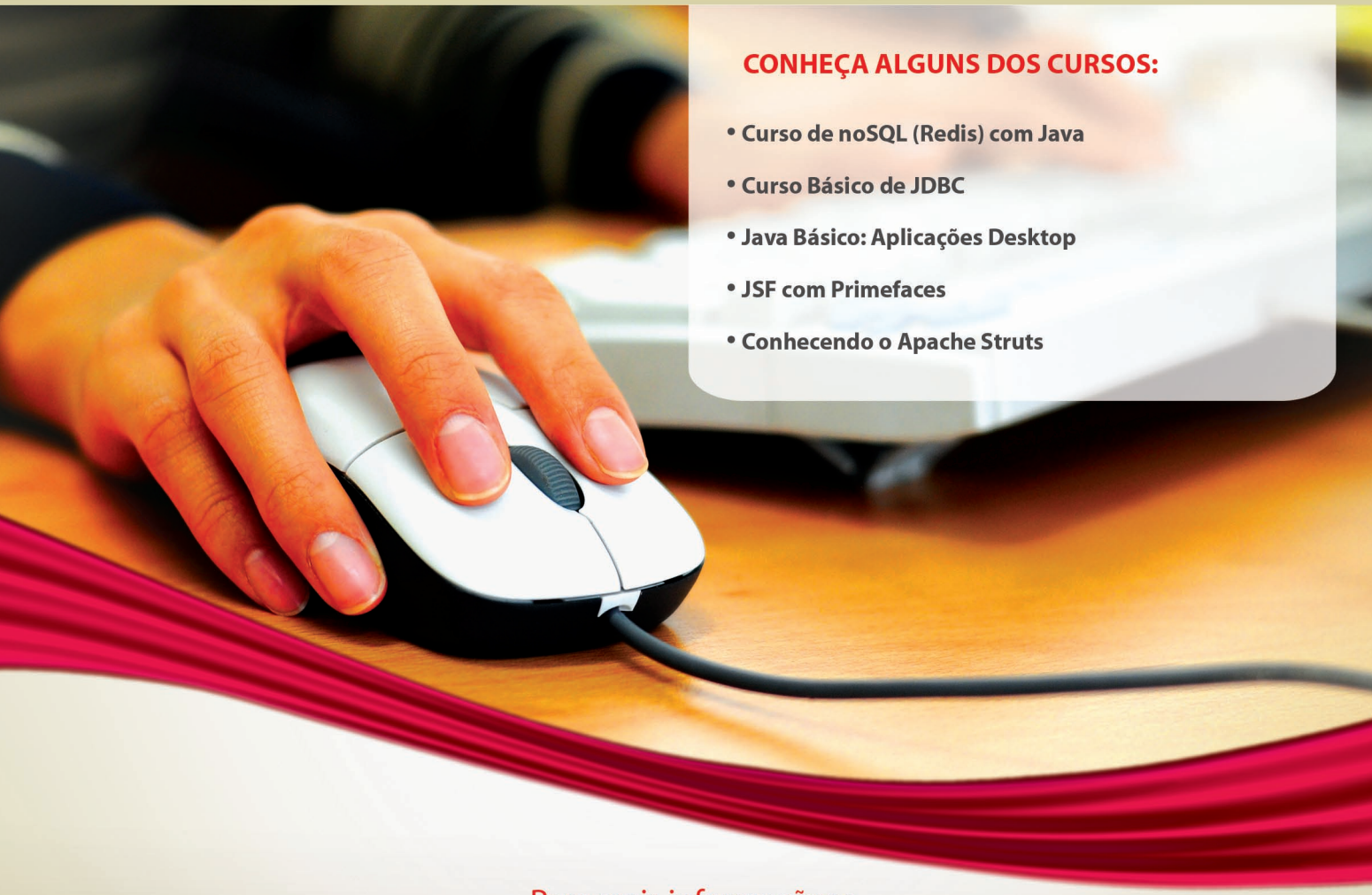


Figura 2. Tela do aplicativo para listagem de impressoras

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

CURSOS ONLINE

A Revista Java Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.



CONHEÇA ALGUNS DOS CURSOS:

- Curso de noSQL (Redis) com Java
- Curso Básico de JDBC
- Java Básico: Aplicações Desktop
- JSF com Primefaces
- Conhecendo o Apache Struts

Para mais informações :

www.devmedia.com.br/curso/javamagazine

(21) 3382-5038

http://www.devmedia.com.br

DEVMEDIA

Listagem 4. Listando impressoras na aplicação

```
01 function TFrm_Principal.ImpreRede(var Str: PChar): PChar;
02 var
03  P: PChar;
04 begin
05  Result := Str;
06  if Str = nil then Exit;
07  P := Str;
08  while P^ = ' ' do Inc(P);
09  Result := P;
10  while (P^ <> #0) and (P^ <> ',') do Inc(P);
11  if P^ = ',' then
12  begin
13   P^ := #0;
14   Inc(P);
15  end;
16  Str := P;
17 end;
18
19 function TFrm_Principal.PegaImpressora: TStrings;
20 var
21  LinAtu, Porta: PChar;
22  Buffer, PrinterInfo: PChar;
23  Flags, Count, NumInfo: DWORD;
24  I: Integer;
25  Level: Byte;
26 begin
27  if ListImpre = nil then
28  begin
29   ListImpre := TStringList.Create;
30   Result   := ListImpre;
31   Try
32    if Win32Platform = VER_PLATFORM_WIN32_NT then
33    begin
34     Flags := PRINTER_ENUM_CONNECTIONS or PRINTER_ENUM_LOCAL;
35     Level := 4;
36    end
37    else
38    begin
39     Flags := PRINTER_ENUM_LOCAL;
40     Level := 5;
41    end;
42    Count := 0;
43    EnumPrinters(Flags, nil, Level, nil, 0, Count, NumInfo);
44    if Count = 0 then Exit;
45    GetMem(Buffer, Count);
46    Try
47     if not EnumPrinters(Flags, nil, Level, PByte(Buffer), Count, Count, NumInfo) then
48      Exit;
49     PrinterInfo := Buffer;
50     for I := 0 to NumInfo - 1 do
51     begin
52      if Level = 4 then
53       with PPrinterInfo4(PrinterInfo)^ do
54       begin
55        ListImpre.AddObject(pPrinterName, Lbx_Impressora);
56        Inc(PrinterInfo, sizeof(TPrinterInfo4));
57       end
58      else
59       with PPrinterInfo5(PrinterInfo)^ do
60       begin
61        LinAtu := pPortName;
62        Porta := ImpreRede(LinAtu);
63        while Porta^ <> #0 do
64        begin
65         ListImpre.AddObject(pPrinterName +''+ Porta, Lbx_Impressora);
66         Porta := ImpreRede(LinAtu);
67        end;
68        Inc(PrinterInfo, sizeof(TPrinterInfo5));
69       end;
70     end;
71    Finally
72     FreeMem(Buffer, Count);
73    end;
74   Except
75    ListImpre.Free;
76    ListImpre := nil;
77    Raise;
78   end;
79  end;
80
81  Result := ListImpre;
82 end;
```

Na seção *uses* é necessário incluir as units WinSpool e Printers. Em seguida, devemos declarar na seção *private* uma variável do tipo TStrings e duas funções que farão a listagem dos dados, da seguinte forma:

```
ListImpre: TStrings;
function PegaImpressora: TStrings;
function ImpreRede(var Str: PChar): PChar;
```

A função ImpreRede, será utilizada para obter as impressoras que estão na rede, já a função PegaImpressora irá listar as impressoras tanto locais como as da rede.

Após a declaração e implementação das funções, basta chamarmos no click do nosso botão, adicionando o resultado ao listbox, da seguinte forma:

```
Lbx_Impressora.Items.AddStrings(PegaImpressora);
```

Reproduzindo Sons

Reproduzir sons em uma aplicação pode ter diversas utilidades, tais como notificar o usuário sobre um evento ou tocar uma música característica da aplicação enquanto um processo é executado, como um arquivo de instruções em áudio.

Função PlaySound

A função PlaySound, declarada na DLL "winmm.dll", permite executar um som no formato Wave, que pode ser inclusive um arquivo embutido na aplicação como recurso. A assinatura dessa função é a que vemos na **Listagem 5.**

O retorno dessa função é 0 se um erro ocorreu, ou um valor diferente de zero se o processo foi bem-sucedido (tocou o som). Os argumentos são os seguintes:

• **lpszName**: O nome ou algum outro identificador do som. O formato exato desse parâmetro vai depender dos valores passados em dwFlags;

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 5. Assinatura da função PlaySound

```
01 (
02   ByVal lpszName as string,
03   ByVal hModule as Long,
04   ByVal dwFlags as Long
05 ) as Long
```

• **hModule**: Um identificador para a aplicação onde está o som a ser tocado (quando for um recurso). Se a função não necessitar desta informação, passe 0 para este parâmetro;
• **dwFlags**: Este argumento recebe zero ou um dos seguintes valores, especificando como lpszName irá tocar o som:
- **SND_ALIAS**: lpszName é uma string identificando o nome do evento do sistema a tocar;
- **SND_ALIAS_ID**: lpszName é uma string com o nome do identificador predefinido para tocar;
- **SND_APPLICATION**: lpszName é uma string identificando a aplicação específica associada ao som a ser tocado;
- **SND_ASYNC**: Essa flag faz com que seja executado um som em modo assíncrono, com o controle retornando para a função assim que o som começa a ser tocado. O som fica então sendo executado em segundo plano;
- **SND_FILENAME**: lpszName é uma string identificando o nome do arquivo .wav para tocar;
- **SND_LOOP**: Esse valor faz tocar o som em um loop até que essa função seja chamada outra vez. Neste caso, SND_ASYNC também deve ser especificado;
- **SND_MEMORY**: Quando esse valor é informado, lpszName representa um ponteiro numérico que aponta para a posição da memória onde está armazenado o som a ser tocado;
- **SND_NODEFAULT**: Se o som especificado não pode ser encontrado, a função é terminada com falha. Se este parâmetro não for especificado, o som SystemDefault é usado se o som especificado não for localizado e a função voltará com sucesso;
- **SND_NOSTOP**: Se um som está tocando, faz o som parar de tocar, ou retorna uma falha se não conseguir parar;
- **SND_NOWAIT**: Se um som já está tocando, não espera o som para de tocar e devolve uma mensagem de erro;
- **SND_PURGE**: Pare a repetição de qualquer som no formato wave. Neste caso lpszName deve ser uma string vazia;
- **SND_RESOURCE**: lpszName é o identificador numérico do som armazenado em um recurso;
- **SND_SYNC**: Executa o som em modo síncrono e não retorna o controle do fluxo para a função até o som ter terminado.

Função Beep

A função Beep está declarada em "kernel32.dll" e sua assinatura pode ser vista na **Listagem 6.**

Essa função reproduz um som padrão do sistema operacional, o qual varia de acordo com a versão do Windows. Com sua chamada sempre é executado o som do sistema SystemDefault, independentemente dos valores passados. Se ocorrer um erro, a função retorna 0 (e neste caso utiliza-se a função GetLastError para obter o código de erro), já se for bem-sucedida, a função retorna um valor diferente de zero. Os argumentos dessa função são bem simples e descritos a seguir.
• **dwFreq**: A frequência, em hertz (Hz), do tom a ser tocado;
• **dwDuration**: A duração, em milissegundos, para tocar o som.

Listagem 6. Assinatura da função Beep

```
01 (
02   ByVal dwFreq As Long,
03   ByVal dwDuration As Long
04 ) As Long
```

Criando uma aplicação para reprodução de sons

Criaremos agora uma aplicação de exemplo onde veremos como utilizar as funções PlaySound e Beep. Inicie uma nova aplicação do tipo VCL Forms e monte sua interface de acordo com a **Figura 3.**

Com a interface pronta, inclua na seção uses a unit MMSystem e altere o código do evento OnClick dos botões de acordo com a **Listagem 7.**

Figura 3. Tela do aplicativo para reprodução de sons

Listagem 7. Código dos botões que reproduzem os sons

```
01 procedure TForm1.Btn_MusicaClick(Sender: TObject);
02 begin
03   PlaySound('C:\WINDOWS\MEDIA\ringin.wav', 0, SND_ASYNC);
04 end;
05
06 procedure TForm1.Btn_BeepClick(Sender: TObject);
07 begin
08   Beep;
09 end;
```

Desenhando Formas

Utilizando a API do Windows também podemos desenhar formas geométricas em nossas aplicações, o que pode ser útil para interfaces não convencionais, que precisem de recursos visuais diferenciados, ou mesmo para desenhar elementos ligados à identidade visual da aplicação.

Função Ellipse

A função Ellipse está declarada em "gdi32.dll" e serve para desenhar uma elipse. Sua assinatura pode ser vista na **Listagem 8.**

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 8. Assinatura da função Ellipse

```
01 (
02   ByVal hdc As Long,
03   ByVal X1 As Long,
04   ByVal Y1 As Long,
05   ByVal X2 As Long,
06   ByVal Y2 As Long
07 ) As Long
```

Os dois pares de coordenadas passados para a função, não são diretamente parte da própria elipse, mas definem os limites de um retângulo no qual a elipse será incluída. A elipse é desenhada com a cor corrente do dispositivo e é preenchida com a cor de preenchimento atual, se houver. A função retorna 0 se falhar, ou 1 se for bem-sucedida. Os argumentos são:

- **hdc**: O contexto do objeto dispositivo para desenhar;
- **X1**: A coordenada X do canto superior esquerdo do retângulo delimitador;
- **Y1**: A coordenada Y do canto superior esquerdo do retângulo delimitador;
- **X2**: A coordenada x do canto inferior direito do retângulo delimitador;
- **Y2**: A coordenada y do canto inferior direito do retângulo delimitador.

Função Rectangle

Também declarada na DLL "gdi32.dll", a função Rectangle desenha um retângulo na tela e sua assinatura e retorno são idênticos aos da função Ellipse.

Função LineTo

A função LineTo também está declarada em "gdi32.dll" e sua assinatura é vista na **Listagem 9.**

Listagem 9. Assinatura da função LineTo

```
01 (
02   ByVal hdc As Long,
03   ByVal x As Long,
04   ByVal y As Long
05 ) As Long
```

Essa função desenha uma linha a partir do ponto atual até o ponto especificado. A linha é desenhada na cor especificada pela propriedade ForeColor desse objeto. Depois que o caminho é traçado, o ponto final é o novo ponto inicial. A função retorna 0 se teve erro, ou um em caso de sucesso.

ShellExecute

Declarada na DLL "shell32.dll", a função ShellExecute tem sua declaração como vemos na **Listagem 10.**

Essa função usa o shell para abrir ou imprimir um arquivo ou ainda executar um programa. Se um programa executável é especificado, o Windows irá executar esse programa. Se um arquivo de documento é especificado, o Windows irá abrir ou imprimi-lo usando o programa associado. Se for bem-sucedida, a função retorna um identificador para a instância do programa aberto ou, no caso de impressão, um identificador para o aplicativo de servidor DDE invocado. Se não tiver êxito, a função retorna 0 (ou seja, falta de memória ou recursos) ou um dos seguintes parâmetros de código de erro:

- **ERROR_FILE_NOT_FOUND**: O arquivo especificado não pôde ser encontrado;
- **ERROR_PATH_NOT_FOUND**: O diretório especificado não pôde ser encontrado;
- **ERROR_BAD_FORMAT**: O arquivo executável especificado (.EXE) foi de alguma forma inválido;
- **SE_ERR_ACCESSDENIED**: Windows negou o acesso ao arquivo especificado;
- **SE_ERR_ASSOCINCOMPLETE**: A associação filename está incompleta ou inválido;
- **SE_ERR_DDEBUSY**: A ação DDE não pôde ocorrer porque outras ações DDE estão em processo;
- **SE_ERR_DDEFAIL**: A transação DDE falhou;
- **SE_ERR_DDETIMEOUT**: A transação DDE não foi concluída porque a solicitação expirou;
- **SE_ERR_DLLNOTFOUND**: O arquivo DLL especificado não foi encontrado;
- **SE_ERR_FNF**: O mesmo que ERROR_FILE_NOT_FOUND;
- **SE_ERR_NOASSOC**: Não há nenhum programa associado com o tipo específico de arquivo;
- **SE_ERR_OOM**: O Windows não tem memória suficiente para executar a operação;
- **SE_ERR_PNF**: O mesmo que ERROR_PATH_NOT_FOUND;
- **SE_ERR_SHARE**: Ocorreu violação no compartilhamento do recurso acessado.

Listagem 10. Assinatura da função ShellExecute

```
01 ShellExecute (
02   ByVal hwnd As Long,
03   ByVal lpOperation As String,
04   ByVal lpFile As String,
05   ByVal lpParameters As String,
06   ByVal lpDirectory As String,
07   ByVal nShowCmd As Long
08 ) As Long
```

Os parâmetros recebidos pela função são descritos a seguir.

- **hwnd**: O identificador da janela que irá chamar a função;
- **lpOperation**: A operação para executar em lpFile. "Open" significa abrir o arquivo ou executar o programa. "Print" significa imprimir o documento;
- **lpFile**: O arquivo para executar a operação;
- **lpParameters**: Todos os parâmetros de linha de comando para passar para um aplicativo aberto;
- **lpDirectory**: O diretório de trabalho para a operação;
- **nshowCmd**: Este argumento recebe um dos seguintes valores, especificando como exibir qualquer janela que se abre na função:

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

- **SW_HIDE**: Esconder a janela aberta;
- **SW_MAXIMIZE**: Maximizar a janela aberta;
- **SW_MINIMIZE**: Minimizar a janela aberta;
- **SW_RESTORE**: Restaurar a janela aberta (não maximizada nem minimizada);
- **SW_SHOW**: Mostrar a janela aberta;
- **SW_SHOWMAXIMIZED**: Mostrar a janela aberta maximizada;
- **SW_SHOWMINIMIZED**: Mostrar a janela aberta minimizada;
- **SW_SHOWMINNOACTIVE**: Mostrar a janela aberta minimizada, mas não ativa;
- **SW_SHOWNA**: Mostrar a janela aberta em seu estado atual, mas não ativá-la;
- **SW_SHOWNOACTIVATE**: Mostrar a janela aberta em seu tamanho e posição mais recente, mas não ativá-la;
- **SW_SHOWNORMAL**: Mostrar a janela aberta e ativá-la (como de costume).

Essa função pode ter diversos usos nos mais variados tipos de aplicações, tais como na abertura de arquivos externos (PDFs, por exemplo) ou execução de aplicações auxiliares para executar tarefas que não podem ser executadas na própria aplicação.

Chamando a ajuda do Windows

A Ajuda do Windows pode ser útil para auxiliar o usuário na execução de alguma tarefa. Esse tipo de ajuda está presente em muitas aplicações e geralmente é acessado a partir da tecla F1, não havendo, porém, regra fixa para definição desse atalho.

Função WinHelp

A função WinHelp está declarada em "user32.dll" e sua função é a que vemos na **Listagem 11.**

Listagem 11. Assinatura da função WinHelp

```
01 WinHelp (
02   ByVal hwndMain As Long,
03   ByVal lpHelpFile As String,
04   ByVal uCommand As Long,
05   dwData As Any
06 ) As Long
```

WinHelp abre um arquivo de Ajuda do Windows, ou manipula o arquivo de ajuda aberto. Se ocorrer um erro, a função retorna 0 (e usa-se GetLastError para obter o código de erro) e se for bem-sucedida a função retorna um valor diferente de zero. Os argumentos que devem ser passados para essa função são os seguintes:

• **hwndMain**: Na maioria dos casos, este argumento representa um identificador para a janela a ser aberta. Se uCommand for passado como HELP_CONTEXTMENU ou HELP_WM_HELP, este é um identificador para um controle específico, para abrir uma ajuda referente ao contexto;

• **lpszHelp**: O nome do arquivo de ajuda para exibir. O nome do arquivo pode ser seguido pelo caractere > e o nome de uma janela de ajuda secundária (definindo o nome do arquivo de ajuda) para abrir, em vez de abrir o primeiro;

• **uCommand**: Esse parâmetro recebe um dos seguintes valores, especificando qual ação a função deverá assumir com o arquivo de ajuda:

- **HELP_COMMAND**: Indica que o argumento dwData possui o identificador de uma macro a ser executada;
- **HELP_CONTENTS**: Exibe o tópico conteúdo do arquivo de ajuda, neste caso dwData deve ser 0. Este parâmetro está obsoleto e deve-se usar o sinalizador HELP_FINDER.
- **HELP_CONTEXT**: Exibir o tópico identificado pelo valor passado como dwData.
- **HELP_CONTEXTMENU**: Exibir o tópico da ajuda associado com o controle selecionado da janela, em uma janela pop-up. Neste caso dwData é uma matriz de pares de Longs (DWords). A primeira parte é um identificador do controle e a segunda é o identificador do contexto do tópico de ajuda associado. Os últimos dados da matriz devem ser dois zeros;
- **HELP_CONTEXTPOPUP**: Exibir o tópico identificado pelo valor passado como dwData em uma janela pop-up;
- **HELP_FINDER**: Exibir a caixa de diálogo de tópicos de ajuda. O argumento dwData deve receber 0 neste caso;
- **HELP_FORCEFILE**: Certifica de que a ajuda do Windows está exibindo o arquivo de ajuda correta; se não for, então irá exibir o correto. O valor de dwData deve ser 0;
- **HELP_HELPONHELP**: Mostrar a ajuda sobre como usar arquivo de ajuda do Windows, que faz parte do Windows. O valor de dwData deve ser 0;
- **HELP_INDEX**: O mesmo que HELP_CONTENTS;
- **HELP_KEY**: Exibir o tópico de ajuda correspondente ao valor passado no argumento dwData. Várias palavras-chave podem ser passadas, separadas por vírgula;
- **HELP_MULTIKEY**: Exibir o tópico especificado por uma palavra-chave em uma tabela de palavras-chaves alternativa. Neste caso dwData deve ser uma estrutura MULTIKEYHELP que especifica um identificador de palavra-chave;
- **HELP_PARTIALKEY**: O mesmo que HELP_KEY, exceto que para exibir o índice sem passar uma palavra-chave, deve-se passar uma string vazia no dwData;
- **HELP_QUIT**: Fechar a ajuda do Windows, a menos que outros programas estejam usando;
- **HELP_SETCONTENTS**: Definir qual tópico de ajuda, é considerado o tema do Conteúdo. DwData é o identificador do contexto do tema, para definir qual será o conteúdo;
- **HELP_SETINDEX**: O mesmo que HELP_SETCONTENTS;
- **HELP_SETPOPUP_POS**: Define a posição de uma janela pop-up subsequente. O parâmetro dwData deve ser uma estrutura POINT_TYPE que identifica as coordenadas do canto superior esquerdo da janela pop-up subsequente;
- **HELP_SETWINPOS**: Exibir a janela de ajuda se estiver minimizada ou oculta, e definir o seu tamanho e posição.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

O dwData deve ser uma estrutura HELPWININFO, que define o tamanho e a posição da janela de ajuda desejado;
- **HELP_TCARD**: Indica que o tema é para mostrar em um cartão. Este valor deve ser combinado com outro parâmetro;
- **HELP_WM_HELP**: Exibir o tópico para o controle identificado por hwndMain. O dwData deve ser uma matriz de pares de Longs (DWords). A primeira parte será um identificador do controle e a segunda o identificador do contexto do tópico de ajuda associado. Os últimos dados da matriz devem ser dois zeros.

• dwData: Esse argumento tem valor variável e depende do valor de uCommand.

Criando uma aplicação para chamar a Ajuda

Aqui desenvolveremos uma aplicação em que utilizaremos diversas combinações de valores para os parâmetros. A interface da aplicação de exemplo pode ser vista na **Figura 4**, onde cada botão fará uma chamada diferente à função WinHelp, conforme mostra a **Listagem 12.**

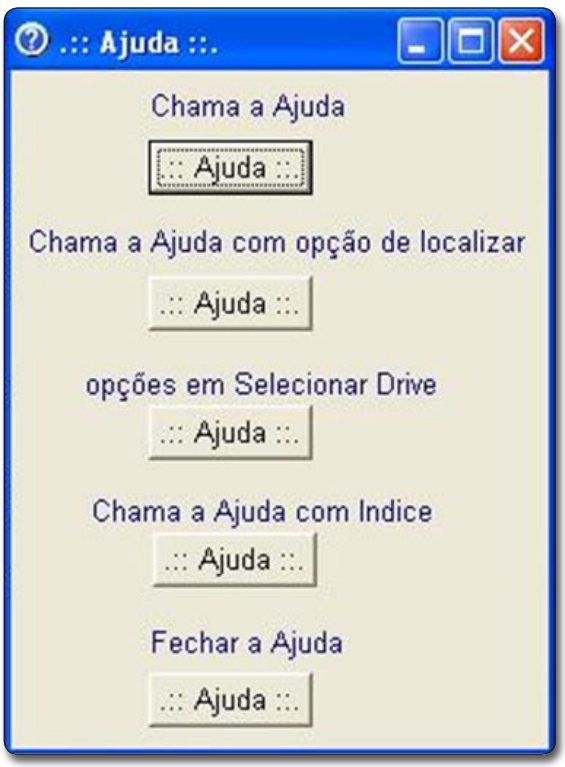


Figura 4. Tela do aplicativo para chamar a Ajuda do Windows

O Windows oferece nativamente diversas funções bastante úteis para a implementação de certas funcionalidades em aplicações, minando a necessidade de desenvolvimento de funções próprias para executar tarefas comuns, como o desenho de formas geométricas e reprodução de sons. Com essas funções podemos otimizar o funcionamento das aplicações e garantir compatibilidade com o sistema operacional, uma vez que estamos utilizando funções nativas contidas em algumas das principais DLLs do Windows.

Listagem 12. Chamando a Ajuda do Windows

```
01 procedure TForm1.BtnAjudaClick(Sender: TObject);
02 var
03   Ajuda: Hwnd;
04 begin
05   Ajuda := FindWindow(Nil, Pchar('windows.hlp'));
06   WinHelp(Ajuda, 'C:\WINDOWS\Help\windows.hlp', HELP_FINDER, 0);
07 end;
08
09 procedure TForm1.BtnLocalizarClick(Sender: TObject);
10 var
11   Ajuda: Hwnd;
12 begin
13   Ajuda := FindWindow(Nil, Pchar('windows.hlp'));
14   WinHelp(Ajuda, 'C:\WINDOWS\Help\windows.hlp', HELP_KEY, 0);
15 end;
16
17 procedure TForm1.BtnDriveClick(Sender: TObject);
18 var
19   Ajuda: Hwnd;
20 begin
21   Ajuda := FindWindow(Nil, Pchar('windows.hlp'));
22   WinHelp(Ajuda, 'C:\WINDOWS\Help\windows.hlp', HELP_CONTENTS, $000F);
23 end;
24
25 procedure TForm1.BtnIndiceClick(Sender: TObject);
26 var
27   Ajuda: Hwnd;
28 begin
29   Ajuda := FindWindow(Nil, Pchar('windows.hlp'));
30   WinHelp(Ajuda, 'C:\WINDOWS\Help\windows.hlp',HELP_PARTIALKEY ,00);
31 end;
32
33 procedure TForm1.BtnFecharClick(Sender: TObject);
34 var
35   Ajuda: Hwnd;
36 begin
37   Ajuda := FindWindow(Nil, Pchar('windows.hlp'));
38   WinHelp(Ajuda, 'C:\WINDOWS\Help\windows.hlp',HELP_QUIT , 128);
39 end;
```


Autor

Vanderson Cavalcante de Freitas
vanderson.freitas@ig.com.br
Analista desenvolvedor Delphi há mais de 5 anos, com experiência em médias e grandes empresas de São Paulo. Formado em técnico em informática no ano de 2003, com diversos cursos em formação específica, como Oracle, Delphi e C#.

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

FÓRUM DEVMEDIA

O lugar perfeito para você ficar por dentro de tudo o que acontece nas tecnologias do mercado atual

No fórum da DevMedia você irá encontrar uma equipe disponível e altamente qualificada com consultores e colaboradores prontos para te ajudar a qualquer hora e sobre qualquer assunto. Temos as salas de Java, .NET, Delphi, Banco de Dados, Engenharia de Software, PHP, Java Script, Web Design, Automação comercial, Ruby on Rails e muito mais!

ACESSE AGORA
www.devmedia.com.br/forum

DEVMEDIA

Desenvolvendo um Sistema Financeiro em Delphi – Parte 4

Finalizando o sistema com a interface do usuário

Este artigo faz parte de um curso

Fique por Dentro

Um sistema bem estruturado, utilizando camadas e as melhores práticas da orientação a objetos, garante diversos benefícios para o processo de desenvolvimento, tais como facilidade de manutenção, flexibilidade e escalabilidade, características essenciais para a maioria dos sistemas construídos atualmente. Nesta última parte da série construiremos a camada final da aplicação, a interface do usuário, que utilizará os serviços das camadas inferiores para exibir e receber as informações em seus controles visuais.

O controle financeiro das empresas é uma das principais áreas, pois depende disso sua sobrevivência frente à economia e seus concorrentes do ramo, por isso a gestão financeira é de extrema importância e podemos encontrá-la em qualquer organização, sendo ela de pequeno, médio ou grande porte.

Para que qualquer empresa mantenha um balanço financeiro favorável, deve tomar decisões estratégicas quase que diariamente, visando expandir os resultados e maximizar os lucros, por isso um sistema financeiro torna-se essencial neste momento, para que as decisões dessa ordem possam ser inteligentes o suficiente para manter a empresa no lucro.

Já vimos que um controle financeiro simples tem basicamente dois módulos: contas a pagar e contas a receber.

Contas a pagar são as obrigações da empresa que devem ser quitadas, de preferência dentro do prazo do vencimento. É de essencial importância para qualquer empresa a quitação de todas as obrigações dentro do prazo acordado, para que esta tenha confiança no mercado e seus fornecedores possam lhe prover matérias primas e produtos através de parcelamentos.

Para que as contas a pagar não caiam no esquecimento, é muito importante que o controle financeiro possua relatórios rápidos que alertem os administradores das finanças sobre as próximas obrigações, com seus totais e as respectivas datas de vencimento. Desta maneira, a empresa pode se antecipar e arrecadar fundos para a quitação das despesas e, por exemplo, procurar efetuar cobranças de clientes, caso seja necessário, ou ainda negociar com seus credores parcelamentos e maiores prazos.

As contas a pagar são adquiridas de diversas formas, mas principalmente na aquisição de matérias primas, contratação de serviços terceirizados e pagamentos de salários, além das despesas básicas para o seu funcionamento, como água, luz, telefone e internet.

Com um bom módulo de contas a pagar conseguimos administrar e evitar diversos problemas, como o pagamento de juros pelo atraso de pagamentos devido a esquecimentos da pessoa responsável. Em caso de problemas financeiros, pode-se antecipar e negociar a dívida com o seu credor, além de evitar problemas com fluxo de caixa.

Já o módulo de contas a receber é, no geral, o controle de vendas que são efetuadas a prazo. Nesse caso, também não importa o tamanho da empresa, vender a prazo traz comodidade e praticidade aos clientes e uma empresa que deseja prosperar no seu ramo de atuações, quase que obrigatoriamente deve oferecer esta opção.

Para que as vendas a prazo não se tornem um problema para a empresa, é muito importante que ela possua um controle eficiente dos seus valores e datas, de maneira que possa identificar os clien-

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

tes que cumprem os pagamentos em dia, para sempre oferecer a eles as melhores condições e ofertas, por exemplo.

Vendas a prazo são feitas de diversas maneiras, sendo a mais comum e segura nos dias de hoje a utilização de cartão de crédito, crediários próprios, que são bastante utilizados no comércio varejista, e boletos bancários com vencimentos mensais.

O controle de contas a receber possibilita ainda o contato com o cliente no caso de atrasos, oferecendo-lhe opções de pagamento. Com essas informações podemos também fazer uma previsão de recebimentos que temos num determinado período de tempo.

Interface Principal

A interface principal do sistema será bastante simples, temos somente um componente da classe *TMainMenu* que terá dois menus, sendo um para cadastro de clientes e fornecedores e outro para o controle de contas a pagar e a receber.

Observe na **Listagem 1,** onde vemos a seção *implementation* do form principal, que optamos por criar os formulários que serão chamados pelos itens do menu em tempo de execução, isto é uma boa prática e otimiza o espaço de memória alocado, o que é bastante recomendável quando possuímos vários formulários.

Para que os forms possam ser localizados, é preciso incluir suas units na seção *uses*, como vemos na linha 5. No evento *OnClick* a cada menu, criamos o form com o método Application.CreateForm, exibimos ele na tela (ShowModal) e em seguida o liberamos da memória (*Free*).

Para que os formulários não sejam criados automaticamente ao iniciarmos a aplicação, devemos configurar no item *Forms* da tela de configurações *Project Options*, conforme mostra a **Figura 1**. Assim, somente o formulário principal da aplicação é criado, ou seja, o frmMain, os demais são criados somente quando forem necessários.

Observe que a criação e liberação dos formulários é idêntica em todos os menus, desta maneira, podemos criar um procedimento genérico que receba uma classe de formulário e uma instância deste e

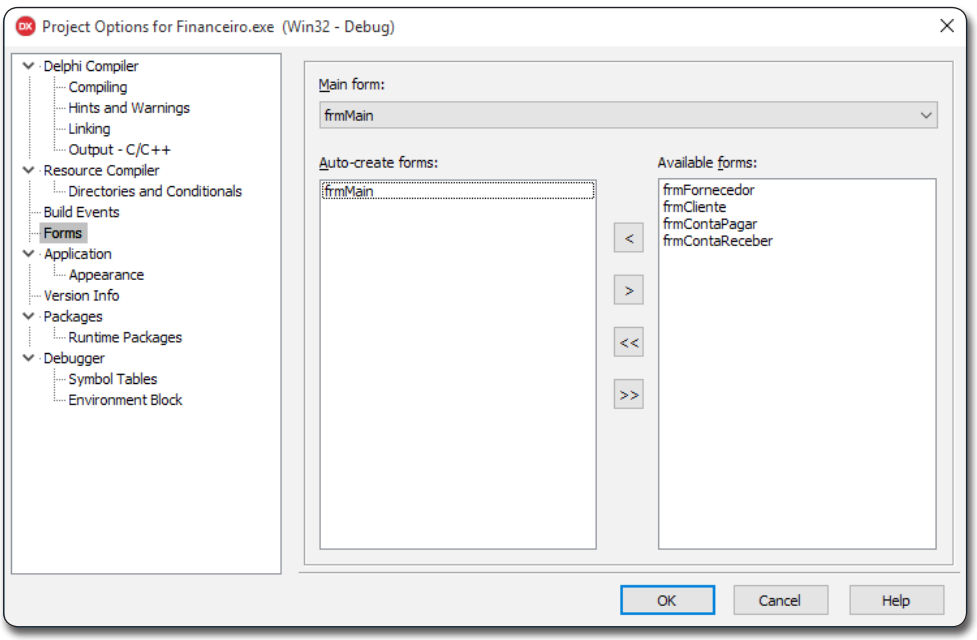


Figura 1. Configuração dos formulários no Project Options

Listagem 1. Implementação do Formulário Principal do Sistema Financeiro

```
01 implementation
02
03 {$R*.dfm}
04
05 uses uModel, uFrmCliente, uFrmFornecedor, uFrmContaPagar, uFrmContaReceber;
06
07 procedure TfrmMain.Cliente1Click(Sender: TObject);
08 begin
09   Application.CreateForm(TfrmCliente, frmCliente);
10   try
11     frmCliente.ShowModal;
12   finally
13     frmCliente.Free;
14   end;
15 end;
16
17 procedure TfrmMain.ContasaPagar1Click(Sender: TObject);
18 begin
19   Application.CreateForm(TfrmContaPagar, frmContaPagar);
20   try
21     frmContaPagar.ShowModal;
22   finally
23     frmContaPagar.Free;
24   end;
25 end;
26
27 procedure TfrmMain.ContasaReceber1Click(Sender: TObject);
28 begin
29   Application.CreateForm(TfrmContaReceber, frmContaReceber);
30   try
31     frmContaReceber.ShowModal;
32   finally
33     frmContaReceber.Free;
34   end;
35 end;
36
37 procedure TfrmMain.Fornecedor1Click(Sender: TObject);
38 begin
39   Application.CreateForm(TfrmFornecedor, frmFornecedor);
40   try
41     frmFornecedor.ShowModal;
42   finally
43     frmFornecedor.Free;
44   end;
45 end;
46
47 end.
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

automaticamente faça a criação e exibição da tela. Na **Listagem 2** vemos como esse método poderia ser implementado.

Listagem 2. Exemplo de procedimento padrão para criação e visualização de formulários

```
01 procedure AbreForm(AClasseForm: TComponentClass; AForm: TForm);
02 begin
03   Application.CreateForm(AClasseForm, AForm);
04   try
05     AForm.ShowModal;
06   finally
07     AForm.Free;
08   end;
09 end;
```

Interfaces para Cadastro de Clientes

No cadastro de clientes possuímos controles visuais para podermos informar o nome, CPF e endereço, além de botões que terão ações específicas para cadastrar um novo cliente, atualizar os dados e limpar os controles de tela. Ainda possuímos um componente da classe *TStringGrid* que servirá para mostrar todos os clientes já cadastrados.

Na **Listagem 3** temos a interface do formulário com os métodos, atributos e componentes utilizados no cadastro de clientes. Já na **Figura 2** temos a interface gráfica final do formulário.

Nas linhas 21 a 23 temos a declaração de três importantes atributos que serão utilizados pela aplicação. O primeiro, chamado *Cliente*, será utilizado para manter uma referência ao cliente que está sendo mostrado na tela a cada troca efetuada no grid,

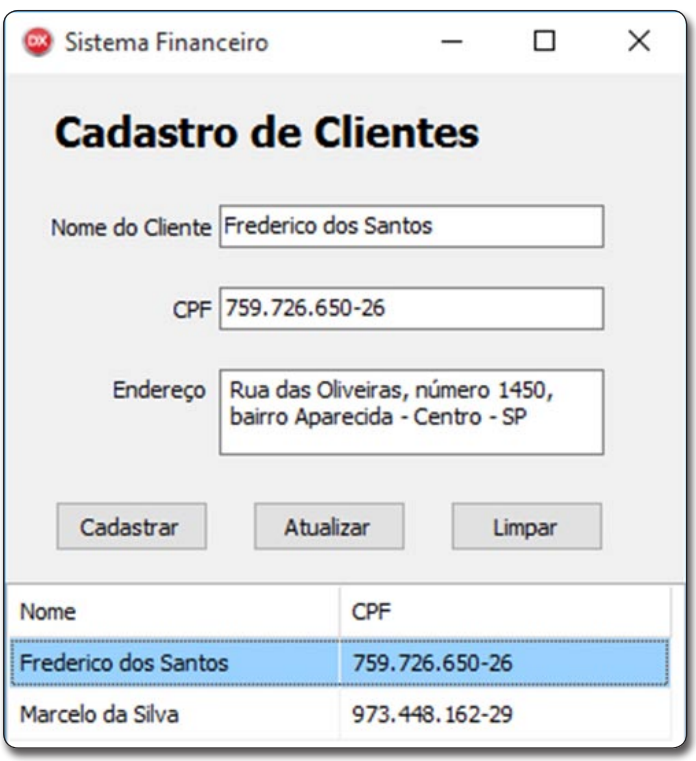


Figura 2. Interface do formulário de Cadastro de Clientes

Listagem 3. Interface do formulário de Cadastro de Clientes

```
01 TfrmCliente = class(TForm)
02   lblAplicacao: TLabel;
03   edtNomeCliente: TLabeledEdit;
04   edtCPFCliente: TLabeledEdit;
05   btnCadastrar: TButton;
06   btnAtualizar: TButton;
07   btnLimpar: TButton;
08   grdClientes: TStringGrid;
09   lblEndereco: TLabel;
10   memEndereco: TMemo;
11   PopupMenu1: TPopupMenu;
12   Excluir1: TMenuItem;
13   procedure FormShow(Sender: TObject);
14   procedure FormClose(Sender: TObject; var Action: TCloseAction);
15   procedure btnLimparClick(Sender: TObject);
16   procedure btnCadastrarClick(Sender: TObject);
17   procedure btnAtualizarClick(Sender: TObject);
18   procedure Excluir1Click(Sender: TObject);
19 private
20   { Private declarations }
21   Cliente: TCliente;
22   Clientes: TObjectList<TCliente>;
23   ClienteDAO: TClienteDAO;
24   procedure ConfigCaptions;
25   procedure AtualizaGrid;
26   procedure PreparaObjeto;
27 public
28   { Public declarations }
29 published
30   procedure PreencheControles(Sender: TObject);
31 end;
```

e também será enviado para persistência (inserção/atualização) com os dados informados pelo usuário nos controles de tela. O segundo, chamado *Clientes*, utiliza *generics* para manter uma lista de clientes, onde serão carregados todos os clientes cadastrados no sistema e serão utilizados para preencher a o *TStringGrid*. Por fim, temos o *ClienteDAO*, que será o mecanismo utilizado para persistir os dados informados pelo usuário no banco de dados e também para carregar os dados já existentes nas tabelas para os objetos. Trabalhando desta forma conseguimos definir camadas lógicas em nosso sistema e evitamos acoplamentos entre as classes.

Aqui é preciso tratarmos os eventos de entrada e saída do formulário (*OnShow* e *OnClose*) para que possamos inicializar alguns dados, e definir o comportamento inicial da tela quando ela é aberta, bem como liberar os recursos quando ela for fechada, conforme podemos observar na **Listagem 4**.

Na abertura do formulário temos a inicialização dos títulos da grade de dados através da chamada do método *ConfigCaptions*, além da criação da instância da classe *TClienteDAO*, que irá permanecer ativa durante toda a ação do usuário dentro do formulário, para que no fim, no evento *OnClose*, seja liberada da memória.

Neste formulário utilizaremos quatro métodos auxiliares para nos ajudar na manipulação de objetos e controles de tela: *ConfigCaptions*, *AtualizaGrid*, *PreparaObjeto*, *PreencheControles*. Podemos ver suas implementações na **Listagem 5**.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 4. Eventos OnShow e OnClose do formulário de cadastro de clientes

```
01 procedure TfrmCliente.FormShow(Sender: TObject);
02 begin
03   ConfigCaptions();
04   Self.ClienteDAO := TClienteDAO.Create;
05   AtualizaGrid();
06 end;
07
08 procedure TfrmCliente.FormClose(Sender: TObject; var Action: TCloseAction);
09 begin
10   Self.ClienteDAO.Free;
11 end;
```

Listagem 5. Métodos auxiliares do formulário de cadastro de clientes

```
01 procedure TfrmCliente.ConfigCaptions;
02 begin
03   grdClientes.Cells[0, 0] := 'Nome';
04   grdClientes.Cells[1, 0] := 'CPF';
05 end;
06
07 procedure TfrmCliente.AtualizaGrid;
08 var
09   ClienteTmp: TCliente;
10   i: Integer;
11 begin
12   i := 1;
13   Self.Clientes := Self.ClienteDAO.FindAll();
14   for ClienteTmp in Self.Clientes do
15   begin
16     grdClientes.Objects[0, i] := ClienteTmp;
17     grdClientes.Cells[0, i] := ClienteTmp.Nome;
18     grdClientes.Cells[1, i] := ClienteTmp.CPF;
19     Inc(i);
20   end;
21   grdClientes.RowCount := i;
22 end;
23
24 procedure TfrmCliente.PreparaObjeto;
25 begin
26   with Self.Cliente do
27   begin
28     Nome := edtNomeCliente.Text;
29     CPF := edtCPFCliente.Text;
30     Endereco := memEndereco.Lines.Text;
31   end;
32 end;
33
34 procedure TfrmCliente.PreencheControles(Sender: TObject);
35 begin
36   Self.Cliente := (grdClientes.Objects[0, grdClientes.Row]) as TCliente;
37   edtNomeCliente.Text := Self.Cliente.Nome;
38   edtCPFCliente.Text := Self.Cliente.CPF;
39   memEndereco.Lines.Text := Self.Cliente.Endereco;
40 end;
```

No método *ConfigCaptions* temos a definição dos rótulos da grade de dados definidos em tempo de execução. Para isso é importante que a propriedade FixedRows esteja definida com o número 1, onde também serão definidos os rótulos.

Já no método *AtualizaGrid* temos primeiramente a busca por todos os clientes do sistema através do método *FindAll*. Depois, através de uma referência temporária à classe *TCliente* e ao utilizarmos o recurso de foreach do Delphi, percorremos todos os itens da lista de clientes e atribuímos o objeto à propriedade objects do *TStringGrid*, que será muito importante, pois é através dela que iremos posteriormente recuperar o objeto selecionado pelo usuário no grid. Também temos a gravação da informação do grid em suas devidas posições. Por fim, apenas definimos o tamanho total da grade através da propriedade *RowCount* do *TStringGrid*.

O método *PreparaObjeto*, da linha 24 a 32, serve para ser invocado antes de uma operação de inclusão ou alteração em banco de dados. Ele prepara uma instância de *TCliente* para ser persistida no banco, coletando os dados informados nos controles de tela pelo usuário e atribuindo-os nas propriedades do objeto *Cliente* do formulário.

Por último, temos o método *PreencheControles*, que difere um pouco dos demais por estar declarado na seção *published* do formulário e possuir um parâmetro *Sender* do tipo *TObject*. O objetivo de termos declarado desta forma é para podermos, em tempo de design, através do *Object Inspector*, atribuirmos diretamente a chamada deste método ao evento *OnClick* do *TStringGrid*, conforme mostra a **Figura 3**. Isso evita que tenhamos que chamar manualmente o método de dentro do evento *OnClick* e a chamada fica referenciada diretamente no arquivo DFM do Delphi.

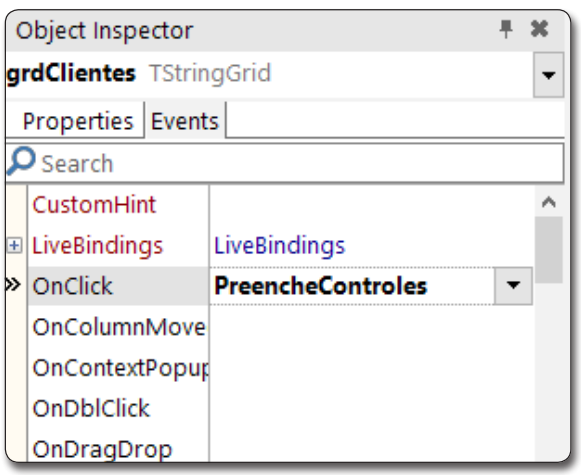


Figura 3. Atribuição de método em evento do StringGrid

Agora veremos as ações executadas pelos três botões que temos em tela (Cadastrar, Atualizar, Limpar), além do *PopupMenu* que temos vinculado ao *TStringGrid* e servirá para excluirmos o cliente que estiver selecionado no momento que o usuário acionar o botão direito do mouse. Veja na **Listagem 6** a implementação do evento *OnClick* de cada um desses controles.

No botão Cadastrar temos primeiramente a criação de uma nova instância da classe *TCliente*, em seguida é invocado o método *PreparaObjeto*, que irá coletar os dados informados pelo usuário nos controles de tela e jogar para as propriedades do objeto, para então invocar o método *Insert* da classe *TClienteDAO*, que irá receber o objeto *TCliente* pronto, com todos as suas propriedades preenchidas que serão persistidas no banco de dados. Por último é chamado o método *AtualizaGrid*, que irá recarregar todos os clientes e já mostrar o cliente recém inserido.

No botão Atualizar já temos um objeto *TCliente* instanciado, porém com os dados em suas propriedades desatualizados em

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

relação aos controles de tela. Por esse motivo, simplesmente é chamado o método *PreparaObjeto*, para que as informações contidas em suas propriedades possam ficar de acordo com o que o usuário informou na interface. Na sequência invocamos o método *Update* da classe *TClienteDAO*, que irá receber o objeto pronto para atualização no banco de dados. Por último é efetuada a atualização da grade de dados para conter os dados atualizados dos clientes e não mostrar informações antigas.

Listagem 6. Ações executadas pelos eventos Cadastrar, Atualizar, Limpar e Excluir

```
01 procedure TfrmFornecedor.btnLimparClick(Sender: TObject);
02 begin
03   edtNomeFornecedor.Clear;
04   edtCNPJForncedor.Clear;
05   memEndereco.Lines.Clear;
06 end;
07
08 procedure TfrmFornecedor.btnCadastrarClick(Sender: TObject);
09 begin
10   Self.Fornecedor := TFornecedor.Create;
11   PreparaObjeto();
12   Self.FornecedorDAO.Insert(Self.Fornecedor);
13   AtualizaGrid();
14 end;
15
16 procedure TfrmFornecedor.btnAtualizarClick(Sender: TObject);
17 begin
18   PreparaObjeto();
19   Self.FornecedorDAO.Update(Self.Fornecedor);
20   AtualizaGrid();
21 end;
22
23 procedure TfrmFornecedor.Excluir1Click(Sender: TObject);
24 begin
25   if MessageDlg('Deseja realmente excluir este fornecedor?', mtConfirmation,
     mbYesNo, 0) = mrYes then
26   begin
27     Self.FornecedorDAO.Delete(TFornecedor(grdFornecedor.Objects
       [0, grdFornecedor.Row]).Id);
28     AtualizaGrid();
29     PreencheControles(Self);
30   end;
31 end;
```

A ação Excluir difere um pouco das demais, primeiramente porque ela é chamada de um menu popup através do *StringGrid*, e também porque possui uma mensagem de confirmação, onde o usuário poderá garantir que deseja realmente excluir aquele registro. Para o método *Delete* da classe *TClienteDAO*, simplesmente passamos o Id, que é a chave primária da tabela CLIENTES, do objeto que está selecionado atualmente na grade de dados através da propriedade *Objects*. Após o registro ser excluído, o *Grid* é atualizado e os controles são novamente preenchidos.

Interfaces para Cadastro de Fornecedores

O formulário de cadastro de fornecedores é bastante semelhante ao cadastro de clientes, basicamente o que diferencia são as referências aos objetos nas linhas 21 a 23, onde ao invés de referenciarmos objetos referentes aos clientes, referenciamos objetos relacionados aos fornecedores, mas que seguem a mesma lógica.

Na **Listagem 7** temos o código do formulário com seus controles e variáveis.

Listagem 7. Interface da classe do formulário de cadastro de fornecedores

```
01 TfrmFornecedor = class(TForm)
02   lblAplicacao: TLabel;
03   edtNomeFornecedor: TLabeledEdit;
04   edtCNPJForncedor: TLabeledEdit;
05   btnCadastrar: TButton;
06   btnAtualizar: TButton;
07   btnLimpar: TButton;
08   grdFornecedor: TStringGrid;
09   lblEndereco: TLabel;
10   memEndereco: TMemo;
11   PopupMenu1: TPopupMenu;
12   Excluir1: TMenuItem;
13   procedure FormShow(Sender: TObject);
14 procedure FormClose(Sender: TObject; var Action: TCloseAction);
15   procedure btnLimparClick(Sender: TObject);
16 procedure btnCadastrarClick(Sender: TObject);
17 procedure btnAtualizarClick(Sender: TObject);
18 procedure Excluir1Click(Sender: TObject);
19 private
20   { Private declarations }
21   Fornecedor: TFornecedor;
22   Fornecedores: TObjectList<TFornecedor>;
23   FornecedorDAO: TFornecedorDAO;
24   procedure ConfigCaptions;
25 procedure AtualizaGrid;
26 procedure PreparaObjeto;
27 public
28   { Public declarations }
29 published
30   procedure PreencheControles(Sender: TObject);
31 end;
```

Na **Figura 4** temos a interface gráfica dessa tela. Observe que devido à semelhança que esta possui com o cadastro de clientes, poderíamos ter criado um formulário base (pai) do qual estas telas poderiam herdar, reduzindo a codificação. Teríamos nesse caso um formulário base (frmBase, por exemplo) e dois filhos que herdariam deste:

- TFrmBase = class(TForm)
- TFrmCliente = class(TFrmBase)
- TFrmFornecedor = class(TfrmBase)

Fica aqui a sugestão para o leitor aplicar essa refatoração, caso ache interessante.

A implementação desse formulário se dá de maneira análoga ao formulário de cadastro de clientes, mudando apenas as classes referenciadas, nomes de controles de telas e nomes de propriedades que diferem de uma classe para outra, conforme evidencia a **Listagem 8**. Os eventos dos botões também poderão ser vistos na listagem.

Manutenção de Contas a Pagar

Para o formulário de contas a pagar, temos a interface definida na **Listagem 9**, com os objetos de negócio *ContaPagar* e *ContaDAO* e seus métodos auxiliares e eventos chamados pelos componentes.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 8. Implementação do formulário de cadastro de fornecedores

```
01 procedure TfrmFornecedor.FormShow(Sender: TObject);
02 begin
03   ConfigCaptions;
04   Self.FornecedorDAO := TFornecedorDAO.Create;
05   AtualizaGrid();
06 end;
07
08 procedure TfrmFornecedor.FormClose(Sender: TObject; var Action: TCloseAction);
09 begin
10   Self.FornecedorDAO.Free;
11 end;
12
13 procedure TfrmFornecedor.ConfigCaptions;
14 begin
15   grdFornecedor.Cells[0, 0] := 'Nome';
16   grdFornecedor.Cells[1, 0] := 'CNPJ ';
17 end;
18
19 procedure TfrmFornecedor.AtualizaGrid;
20 var
21   FornecedorTmp: TFornecedor;
22   i: Integer;
23 begin
24   i := 1;
25   Self.Fornecedores := Self.FornecedorDAO.FindAll();
26   for FornecedorTmp in Self.Fornecedores do
27   begin
28     grdFornecedor.Objects[0, i] := FornecedorTmp;
29     grdFornecedor.Cells[0, i] := FornecedorTmp.Nome;
30     grdFornecedor.Cells[1, i] := FornecedorTmp.CNPJ;
31     Inc(i);
32   end;
33   grdFornecedor.RowCount := i;
34 end;
35
36 procedure TfrmFornecedor.PreencheControles(Sender: TObject);
37 begin
38   Self.Fornecedor := (grdFornecedor.Objects[0, grdFornecedor.Row]) as TFornecedor;
39   edtNomeFornecedor.Text := Self.Fornecedor.Nome;
40   edtCNPJForncedor.Text := Self.Fornecedor.CNPJ;
41   memEndereco.Lines.Text := Self.Fornecedor.Endereco;
42 end;
43
44 procedure TfrmFornecedor.PreparaObjeto;
45 begin
46   with Self.Fornecedor do
47   begin
48     Nome := edtNomeFornecedor.Text;
49     CNPJ := edtCNPJForncedor.Text;
50     Endereco := memEndereco.Lines.Text;
51   end;
52 end;
53
54 procedure TfrmFornecedor.btnLimparClick(Sender: TObject);
55 begin
56   edtNomeFornecedor.Clear;
57   edtCNPJForncedor.Clear;
58   memEndereco.Lines.Clear;
59 end;
60
61 procedure TfrmFornecedor.btnCadastrarClick(Sender: TObject);
62 begin
63   Self.Fornecedor := TFornecedor.Create;
64   PreparaObjeto();
65   Self.FornecedorDAO.Insert(Self.Fornecedor);
66   AtualizaGrid();
67 end;
68
69 procedure TfrmFornecedor.btnAtualizarClick(Sender: TObject);
70 begin
71   PreparaObjeto();
72   Self.FornecedorDAO.Update(Self.Fornecedor);
73   AtualizaGrid();
74 end;
75
76 procedure TfrmFornecedor.Excluir1Click(Sender: TObject);
77 begin
78   if MessageDlg('Deseja realmente excluir este fornecedor?', mtConfirmation,
       mbYesNo, 0) = mrYes then
79   begin
80     Self.FornecedorDAO.Delete(TFornecedor(grdFornecedor.Objects
       [0, grdFornecedor.Row]).Id);
81     AtualizaGrid();
82     PreencheControles(Self);
83   end;
84 end;
```

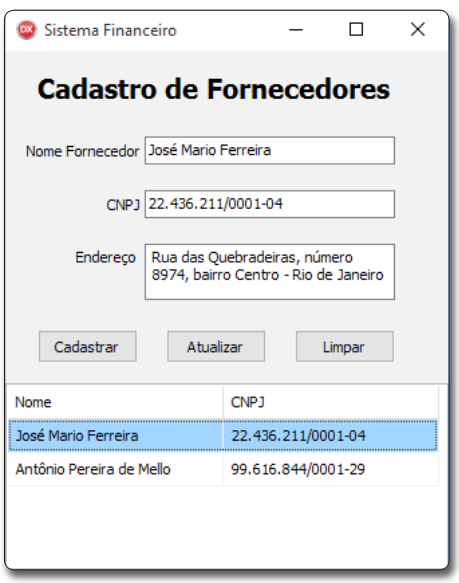


Figura 4. Interface do formulário de Cadastro de Fornecedores

O objeto *ContaPagar* tem a função de receber os dados do banco de dados e exibir para o usuário, bem como receber os dados do usuário e transmitir para a classe *ContaDAO*, que irá fazer a persistência em banco de dados.

Na **Listagem 10** temos a implementação dos procedimentos auxiliares do formulário que controlam os componentes e os dados que irão ficar visível em tela, além de preparar os objetos de negócio para persistência.

O método *CarregarDados* tem a função de carregar todos os itens da conta que foi selecionada na pesquisa. Aqui usamos novamente uma referência temporária a *TItemConta* para percorrer todos os itens da lista através de um laço do tipo foreach e preencher cada linha do *StringGrid*.

Os métodos *ConfigCaptions* (linha 24) e *LimpaControles* (linha 32) são bastante simples, o primeiro apenas configura a descrição de cada coluna da grade de dados e o segundo limpa todas as informações dos controles de tela, para que novas informações possam ser cadastradas.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 9. Interface da manutenção do formulário contas a pagar

```
01  TfrmContaPagar = class(TForm)
02    btnLocalizar: TButton;
03    lblAplicacao: TLabel;
04    btnNova: TButton;
05    rgpSituacao: TRadioGroup;
06    lblFornecedor: TLabel;
07    cbxFornecedor: TComboBox;
08    StaticText1: TStaticText;
09    grdItensConta: TStringGrid;
10    edtValorTotal: TLabeledEdit;
11    edtParcelas: TLabeledEdit;
12    btnSalvar: TButton;
13    btnCancelar: TButton;
14    StaticText2: TStaticText;
15    btnAtualizar: TBitBtn;
16    procedure btnLocalizarClick(Sender: TObject);
17    procedure btnSalvarClick(Sender: TObject);
18    procedure FormCreate(Sender: TObject);
19    procedure FormShow(Sender: TObject);
20    procedure btnNovaClick(Sender: TObject);
21    procedure btnAtualizarClick(Sender: TObject);
22    procedure btnCancelarClick(Sender: TObject);
23  private
24    { Private declarations }
25    ContaPagar: TContaPagar;
26    ContaDAO: TContaDAO;
27    procedure CarregarDados();
28    procedure PreparaObjeto();
29    procedure ConfigCaptions();
30    procedure LimpaControles();
31  public
32    { Public declarations }
33  end;
```

Listagem 10. Métodos auxiliares do formulário de contas a pagar

```
01 procedure TfrmContaPagar.CarregarDados;
02 var
03   ItemContaTmp: TItemConta;
04   i: Integer;
05   aSituacao: string;
06 begin
07   i := 1;
08   for ItemContaTmp in Self.ContaPagar.ItensConta do
09   begin
10     case Self.ContaPagar.Situacao of
11       stAberta: aSituacao := 'Aberta';
12       stPaga: aSituacao := 'Paga';
13       stCancelada: aSituacao := 'Cancelada';
14     end;
15     grdItensConta.Objects[0, i] := ItemContaTmp;
16     grdItensConta.Cells[0, i] := DateToStr(ItemContaTmp.Vencimento);
17     grdItensConta.Cells[1, i] := FloatToStr(ItemContaTmp.Valor);
18     grdItensConta.Cells[2, i] := aSituacao;
19     Inc(i);
20   end;
21   grdItensConta.RowCount := i;
22 end;
23
24 procedure TfrmContaPagar.ConfigCaptions;
25 begin
26   grdItensConta.Cells[0, 0] := 'Parcela';
27   grdItensConta.Cells[1, 0] := 'Vencimento';
28   grdItensConta.Cells[2, 0] := 'Valor';
29   grdItensConta.Cells[3, 0] := 'Situação';
30 end;
31
32 procedure TfrmContaPagar.LimpaControles;
33 begin
34   grdItensConta.RowCount := 1;
35   cbxFornecedor.ItemIndex := -1;
36   edtValorTotal.Clear;
37   edtParcelas.Clear;
38   rgpSituacao.ItemIndex := -1;
39 end;
40
41 procedure TfrmContaPagar.PreparaObjeto;
42 var
43   aSituacao: TSituacao;
44 begin
45   with Self.ContaPagar do
46   begin
47     case rgpSituacao.ItemIndex of
48       0: aSituacao := stAberta;
49       1: aSituacao := stPaga;
50       2: aSituacao := stCancelada;
51     end;
52     Self.ContaPagar.Fornecedor := TFornecedor(cbxFornecedor.Items.
        Objects[cbxFornecedor.ItemIndex]);
53     Self.ContaPagar.Valor := StrToFloatDef(edtValorTotal.Text, 0);
54     Self.ContaPagar.NumParcelas := StrToIntDef(edtParcelas.Text, 1);
55   end;
56 end;
```

O método *PreparaObjeto* deve ser chamado antes de fazermos a persistência dos dados, para que sejam lidos dos controles de tela todos os dados necessários e preenchidas todas as propriedades necessárias para persistência. Veja que na linha 52 fazemos um type casting para garantirmos a compatibilidade entre os objetos.

Na **Listagem 11** temos o evento *OnShow* (linha 54) com a criação da instância da classe *TContaDAO*, que será utilizada enquanto o formulário estiver em execução e liberada somente no evento *OnClose* (linha 60), ou seja, quando o usuário fechar este formulário. No *OnShow* ainda temos a chamada ao procedimento que configura a descrição da grade de dados.

No evento localizar, a partir da linha 12, temos a criação e exibição do formulário de pesquisa de contas a pagar. Após o usuário fechar a interface de pesquisa, temos um teste onde verificamos se o usuário realmente selecionou uma conta, para aí sim atribuí-la à referência *ContaPagar* do formulário principal. Ainda são recarregados os dados de tela para refletir o objeto recém selecionado pelo usuário.

O evento atualizar pode ser executado após o usuário preencher as informações básicas da conta, e a partir dessas informações fazer uma simulação das parcelas, valores e datas de vencimento que serão geradas. Se tudo estiver de acordo, este então pode executar a ação de salvar que irá finalmente persistir as informações em banco de dados.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 11. Implementação dos eventos do formulário de manutenção contas a pagar

```
01 procedure TfrmContaPagar.btnAtualizarClick(Sender: TObject);
02 begin
03   Self.ContaPagar.SimularParcelas();
04   CarregarDados();
05 end;
06
07 procedure TfrmContaPagar.btnCancelarClick(Sender: TObject);
08 begin
09   LimpaControles();
10 end;
11
12 procedure TfrmContaPagar.btnLocalizarClick(Sender: TObject);
13 begin
14   Application.CreateForm(TfrmLocalizarContaPagar, frmLocalizarContaPagar);
15   try
16     frmLocalizarContaPagar.ShowModal;
17     if frmLocalizarContaPagar.ContaPagar <> nil then
18     begin
19       Self.ContaPagar := frmLocalizarContaPagar.ContaPagar;
20       CarregarDados();
21     end;
22   finally
23     frmLocalizarContaPagar.Free;
24   end;
25 end;
26
27 procedure TfrmContaPagar.btnNovaClick(Sender: TObject);
28 begin
29   LimpaControles();
30 end;
31
32 procedure TfrmContaPagar.btnSalvarClick(Sender: TObject);
33 begin
34   Self.ContaPagar := TContaPagar.Create;
35   PreparaObjeto();
36   Self.ContaDAO.Insert(Self.ContaPagar);
37   CarregarDados();
38 end;
39
40 procedure TfrmContaPagar.Cancelar1Click(Sender: TObject);
41 begin
42   Self.ContaDAO.AlteraSituacaoParcela(
43     TItemConta(grdItensConta.Objects[0, grdItensConta.Row]), stCancelada);
44   CarregarDados();
45 end;
46
47 procedure TfrmContaPagar.Quitar1Click(Sender: TObject);
48 begin
49   Self.ContaDAO.AlteraSituacaoParcela(
50     TItemConta(grdItensConta.Objects[0, grdItensConta.Row]), stPaga);
51   CarregarDados();
52 end;
53
54 procedure TfrmContaPagar.FormShow(Sender: TObject);
55 begin
56   Self.ContaDAO := TContaDAO.Create;
57   ConfigCaptions();
58 end;
59
60 procedure TfrmContaPagar.FormClose(Sender: TObject; var Action: TCloseAction);
61 begin
62   Self.ContaDAO.Free;
63 end;
```

O botão Nova apenas limpa os controles para que o usuário possa iniciar um novo cadastro, assim como o Cancelar.

Os dois eventos do *PopupMenu* do *StringGrid* servem para fazer a quitação ou cancelamento de determinada parcela, para isso chamam o método *AlteraSituacaoParcela*, passando qual a conta selecionada atualmente no grid e qual a situação desejada, para posteriormente recarregar os dados atualizados.

Na **Figura 5** temos a interface com dados em tela, onde podemos ver como ela ficou estruturada graficamente.

Agora precisamos implementar o formulário de pesquisa, cujo código se encontra na **Listagem 12**. Observe que temos uma propriedade pública chamada *ContaPagar* que irá receber o objeto de *TContaPagar* pesquisado e selecionado pelo usuário, e que por ser público, ficará visível ao formulário principal de manutenção de contas a pagar.

Na **Figura 6** temos a interface desse formulário, observe que foram utilizados um componente do tipo radio para a situação, um combo para listar os fornecedores disponíveis e uma grade para exibição dos resultados. A pesquisa é efetuada no momento que cada controle é alterado, por isso o método *CarregaGridContas* (linha 21), foi definido na seção *published* com um parâmetro *Sender* do tipo *TObject*, para que possamos fazer o link direto dele com os eventos dos componentes.

No evento *OnDblClick* do grid (linha 46), obtemos o objeto selecionado e o atribuímos a *ContaPagar*, que será lida posteriormente pelo form principal.

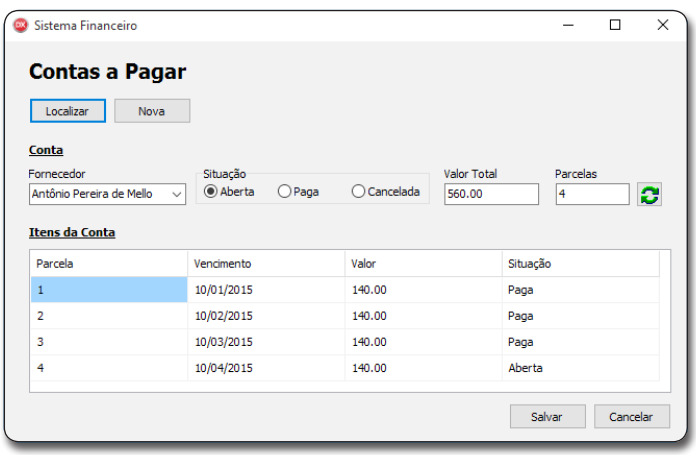


Figura 5. Interface Principal do Contas a Pagar

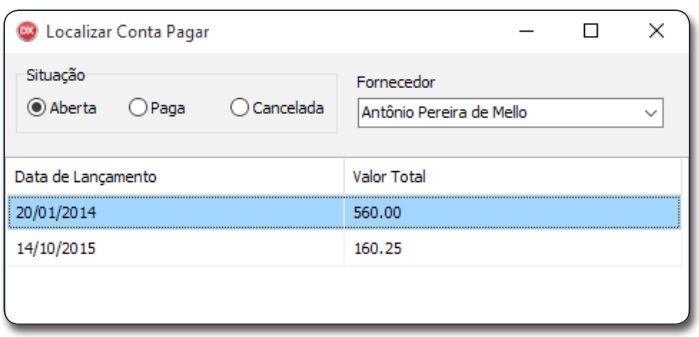


Figura 6. Localizando uma Conta a Pagar

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Manutenção de Contas a Receber

A manutenção do formulário contas a receber é muito semelhante à de contas a pagar, diferenciando apenas na manipulação da classe *TCliente* ao invés da classe *TFornecedor*, como podemos observar na **Listagem 13**.

Como a maior parte do código segue a mesma lógica da tela de contas a pagar, vemos nessa listagem apenas os pontos principais da implementação dessa tela. Na linha 1 temos o evento *OnClick* do botão Atualizar, que simula as parcelas da conta e exibe os dados na tela. Já o botão Localizar, na linha 7, abre o formulário

Listagem 12. Interface e implementação do localizar contas a pagar

```
01 TfrmLocalizarContaPagar = class(TForm)
02   grdContasPagar: TStringGrid;
03   rgpSituacao: TRadioGroup;
04   cbxCliente: TComboBox;
05   lblFornecedor: TLabel;
06   procedure grdContasPagarDblClick(Sender: TObject);
07   procedure FormShow(Sender: TObject);
08 private
09   { Private declarations }
10   FContaPagar: TContaPagar;
11   Contas: TObjectList<TConta>;
12   ContaDAO: TContaDAO;
13   procedure ConfigCaptions;
14 public
15   { Public declarations }
16   property ContaPagar: TContaPagar read FContaPagar write FContaPagar;
17 published
18   procedure CarregaGridContas(Sender: TObject);
19 end;
20
21 procedure TfrmLocalizarContaPagar.CarregaGridContas(Sender: TObject);
22 var
23   aSituacao: TSituacao;
24 begin
25   if rgpSituacao.ItemIndex = 0 then
26     aSituacao := stAberta
27   else if rgpSituacao.ItemIndex = 1 then
28     aSituacao := stPaga
29   else aSituacao := stCancelada;
30   Self.Contas := Self.ContaDAO.FindAll(aSituacao,
31     TCliente(cbxCliente.Items.Objects[cbxCliente.ItemIndex]), FContaPagar);
32 end;
33
34 procedure TfrmLocalizarContaPagar.ConfigCaptions;
35 begin
36   grdContasPagar.Cells[0, 0] := 'Data de Lançamento';
37   grdContasPagar.Cells[1, 0] := 'Valor Total';
38 end;
39
40 procedure TfrmLocalizarContaPagar.FormShow(Sender: TObject);
41 begin
42   CarregaGridContas(Self);
43   ConfigCaptions();
44 end;
45
46 procedure TfrmLocalizarContaPagar.grdContasPagarDblClick(Sender: TObject);
47 begin
48   Self.ContaPagar := TContaPagar(grdContasPagar.Objects[0, grdContasPagar.Row]);
49 end;
```

Listagem 13. Interface e implementação do formulário de contas a receber

```
01 procedure TfrmContaReceber.btnAtualizarClick(Sender: TObject);
02 begin
03   Self.ContaReceber.SimularParcelas();
04   CarregarDados();
05 end;
06
07 procedure TfrmContaReceber.btnLocalizarClick(Sender: TObject);
08 begin
09   Application.CreateForm(TfrmLocalizarContaReceber, frmLocalizarContaReceber);
10   try
11     frmLocalizarContaReceber.ShowModal;
12     if frmLocalizarContaReceber.grdContasReceber<> nil then
13     begin
14       Self.ContaReceber := frmLocalizarContaReceber.ContaReceber;
15       CarregarDados();
16     end;
17   finally
18     frmLocalizarContaReceber.Free;
19   end;
20 end;
21
22 procedure TfrmContaReceber.btnSalvarClick(Sender: TObject);
23 begin
24   Self.ContaReceber := TContaReceber.Create;
25   PreparaObjeto();
26   Self.ContaDAO.Insert(Self.ContaReceber);
27   CarregarDados();
28 end;
29
30 procedure TfrmContaReceber.Cancelar1Click(Sender: TObject);
31 begin
32   Self.ContaDAO.AlteraSituacaoParcela(
33     TItemConta(grdItensConta.Objects[0, grdItensConta.Row]), stCancelada);
34   CarregarDados();
35 end;
36
37 procedure TfrmContaReceber.Quitar1Click(Sender: TObject);
38 begin
39   Self.ContaDAO.AlteraSituacaoParcela(
40     TItemConta(grdItensConta.Objects[0, grdItensConta.Row]), stPaga);
41   CarregarDados();
42 end;
43
44 procedure TfrmContaReceber.CarregarDados;
45 var
46   ItemContaTmp: TItemConta;
47   i: Integer;
48   aSituacao: string;
49 begin
50   i := 1;
51   for ItemContaTmp in Self.ContaReceber.ItensConta do
52   begin
53     case Self.ContaReceber.Situacao of
54       stAberta: aSituacao := 'Aberta';
55       stPaga: aSituacao := 'Paga';
56       stCancelada: aSituacao := 'Cancelada';
57     end;
58     grdItensConta.Objects[0, i] := ItemContaTmp;
59     grdItensConta.Cells[0, i] := DateToStr(ItemContaTmp.Vencimento);
60     grdItensConta.Cells[1, i] := FloatToStr(ItemContaTmp.Valor);
61     grdItensConta.Cells[2, i] := aSituacao;
62     Inc(i);
63   end;
64   grdItensConta.RowCount := i;
65 end;
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

de pesquisa de contas a receber e caso o usuário selecione uma conta nessa tela, a atribui ao objeto *ContaReceber* (linha 14). Na linha 22 temos a ação de Salvar, que cria um novo objeto do tipo *TContaReceber*, o preenche com os dados da tela e persiste no banco usando o objeto *DAO*. Os menus Cancelar e Quitar, cujo clique é tratado nas linhas 30 e 37, alteram a situação da conta selecionada para stCancelada e stQuitada, respectivamente, e em seguida atualizam os dados na tela. Por fim temos o método *CarregarDados*, (linha 44) que itera sobre os itens da conta a serem exibidos e os adiciona ao grid.

Na **Figura 7** temos a interface desse formulário, que difere da tela de contas a pagar apenas pelo fato de listar clientes em vez de fornecedores.

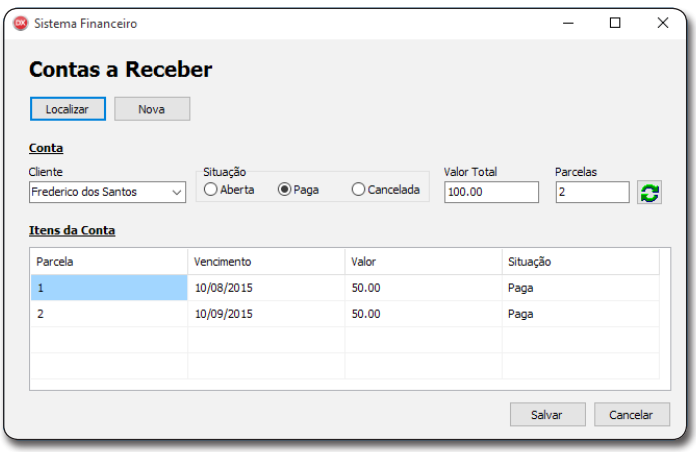


Figura 7. Interface gráfica do formulário de contas a receber

Por fim, resta apenas implementar o formulário de pesquisa de contas a receber, cujo código se encontra na Listagem 14.

Nesse formulário, o objeto público ContaReceber, definido na linha 16 será preenchido com a conta que for selecionada pelo usuário (linha 46) e poderá ser acessado pelo form anterior quando essa tela for fechada. Na linha 21 temos o método *CarregarGridContas*, que utiliza o objeto *DAO* para listar as contas a receber de acordo com a situação e cliente selecionados pelo usuário. A interface gráfica dessa tela pode ser vista na **Figura 8.**

Existem várias formas interessantes de trabalharmos com os componentes da VCL de maneira orientada a objetos. Componentes como o *TStringGrid* e o *TComboBox* possuem a propriedade *Objects*, que permite que façamos um link entre nossas listas de objetos com os componentes de tela, facilitando muito na hora

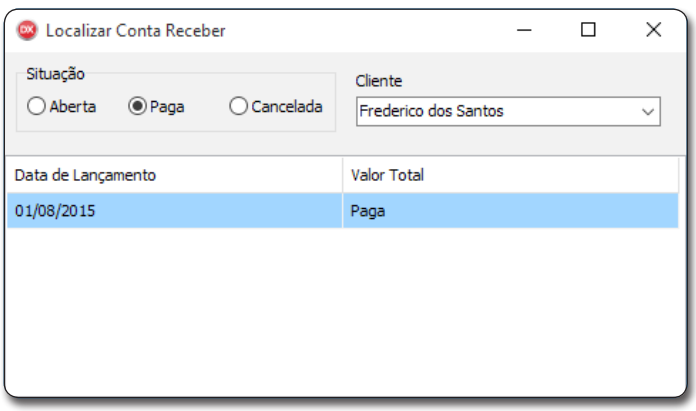


Figura 8. Tela de pesquisa de contas a receber

Listagem 14. Interface e implementação do localizar contas a receber

```
01 TfrmLocalizarContaReceber = class(TForm)
02   grdContasReceber: TStringGrid;
03   rgpSituacao: TRadioGroup;
04   cbxCliente: TComboBox;
05   lblCliente: TLabel;
06   procedure grdContasReceberDblClick(Sender: TObject);
07   procedure FormShow(Sender: TObject);
08 private
09   { Private declarations }
10   FContaReceber: TContaReceber;
11   Contas: TObjectList<TConta>;
12   ContaDAO: TContaDAO;
13   procedure ConfigCaptions;
14 public
15   { Public declarations }
16   property ContaReceber: TContaReceber read FContaReceber
       write FContaReceber;
17 published
18   procedure CarregaGridContas(Sender: TObject);
19 end;
20
21 procedure TfrmLocalizarContaReceber.CarregaGridContas(Sender: TObject);
22 var
23   aSituacao: TSituacao;
24 begin
25   if rgpSituacao.ItemIndex = 0 then
26     aSituacao := stAberta
27   else if rgpSituacao.ItemIndex = 1 then
28     aSituacao := stPaga
29   else aSituacao := stCancelada;
30   Self.Contas := Self.ContaDAO.FindAll(aSituacao,
31     TCliente(cbxCliente.Items.Objects[cbxCliente.ItemIndex]), FContaReceber);
32 end;
33
34 procedure TfrmLocalizarContaReceber.ConfigCaptions;
35 begin
36   grdContasReceber.Cells[0, 0] := 'Data de Lançamento';
37   grdContasReceber.Cells[1, 0] := 'Valor Total';
38 end;
39
40 procedure TfrmLocalizarContaReceber.FormShow(Sender: TObject);
41 begin
42   CarregaGridContas(Self);
43   ConfigCaptions();
44 end;
45
46 procedure TfrmLocalizarContaReceber.grdContasReceberDblClick(Sender: TObject);
47 begin
48   Self.ContaReceber := TContaReceber(grdContasReceber.Objects
       [0, grdContasReceber.Row]);
49 end;
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

em que necessitamos recuperar objetos de domínio selecionados em tela.

Poderíamos otimizar ainda mais a estrutura de nosso projeto caso aplicássemos algum padrão relacionado às interfaces com o usuário, como o padrão de projeto MVVM (**BOX 1**).

BOX 1. Padrão MVVM

MVVM é a abreviatura para Model-View-ViewModel, cuja finalidade principal, em termos gerais, é favorecer a segmentação do contexto da aplicação, dividindo-a em partes bem estabelecidas. Tal separação tende a ser clara o suficiente a ponto de contribuir em aspectos que vão da elaboração da aplicação, passando por sua manutenção e eventual expansão. Na prática, a adoção do padrão MVVM se reflete essencialmente no uso de três classes distintas, que acabam por representar cada um dos elementos citados anteriormente, que compõem sua estrutura:

• Model: diz respeito ao modelo da aplicação, encapsulando toda a lógica de negócio e dados envolvida;

• View: é relativa à interface de usuário, dessa forma, ela ficará responsável por encapsular toda a lógica pertinente à UI;

• ViewModel: pode ser considerada o artefato peculiar do padrão MVVM, que o diferencia dos demais padrões existentes, uma vez que cuida dos aspectos de estado e lógica de apresentação pertinentes.

Inicialmente observamos a necessidade de escrever uma maior quantidade de código para desenvolver uma aplicação dessa maneira, separada em camadas e com responsabilidades bem divididas. Porém, os benefícios dessa abordagem logo surgem e podem ser observados durante todo o ciclo de vida da aplicação, pois diversos aspectos como testes e manutenção se tornam bem mais simples, além de facilitar a expansão e reaproveitamento do código existente, mantendo sempre uma boa organização na estrutura do projeto.

Autor

FILIPE DALEPIANE
filipe.dalepiane@gmail.com
Bacharel em Ciência da Computação, certificado Delphi Developer, colunista das revistas Clube Delphi e .NET Magazine. Trabalha atualmente como Analista de Sistemas na AVMB Consultoria e Assessoria em Informática em Santa Maria – RS.

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

CURSOS ONLINE

.net magazine

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

MAIS de 25 CURSOS DISPONÍVEIS

CONHEÇA OS CURSOS MAIS RECENTES:

- Curso Padrões de Projeto com C#
- Curso básico de ASP .NET
- Curso de Introdução ao .NET Framework
- Curso Básico de C#
- C# 5 e suas novidades
- ASP.NET MVC – Sistema de Vestibular

Para mais informações :

www.devmedia.com.br/curso/netmagazine

(21) 3382-5038

http://www.devmedia.com.br

DEVMEDIA

Como criar um instalador de componentes

Crie um instalador de componentes pelo Delphi

Ao trabalhar com muitos componentes, começamos a ter dificuldades na hora de reinstalá-los devido à grande quantidade e às vezes incompatibilidades entre versões. Esse tipo de tarefa acaba por tomar tempo que poderia ser utilizado para atividades práticas de desenvolvimento.

Seguir os passos tradicionais para instalação de componentes não é uma tarefa fácil nem mesmo para programadores mais experientes, é um processo cansativo e programadores novatos não têm a experiência adequada para tal processo. Componentes como JEDI, DevExpress, FastReport e ACBR têm uma coleção muito grande de pacotes principais e um número muito grande de pacotes dependentes, fazendo com que a instalação se torne mais cansativa, pois deve ser seguida uma ordem, que se não for respeitada pode acarretar erros para o projeto.

Como são distribuídos os componentes

Os componentes no Delphi podem ser distribuídos através da instalação de arquivos de extensão pas, dcu, dpk ou através de instaladores.

• **PAS**: Para arquivos pas, execute o Delphi e feche o projeto, acesse o menu "Component" e clique na opção "Install componente". Na janela que se apresenta, acesse a aba "Into New Packages", clique no botão "Browse" ao lado da caixa de texto "Unit File Name" e abra o arquivo com extensão *.pas. Dê OK e logo após "Compile" e "Install" e o arquivo criará uma aba na barra de componentes com um nome para a sua localização.

• **DCU**: Para arquivos com a extensão *.dcu, o processo é um pouco mais complicado. Acesse o menu "Component" e clique na opção "Install package". Verifique se na lista "Design packages" existe a opção "Borland user componente", se sim, clique no botão "Edit" e será aberta uma caixa de mensagens. Clique no botão "Yes". Na janela que aparece, clique no botão "Add" e na janela que se abrirá clique no botão "Browse" da caixa de texto "Unit file name". Na caixa de combinação "Files of type" escolha "Delphi compiled unit(*.dcu)" e depois na caixa de texto "File name" direcione o arquivo a ser instalado clicando no botão "Open". Clique no botão "Ok" na janela que aparece e clique no botão "Install".

• **DPK**: Para instalar pacotes de componentes, execute o Delphi e feche o projeto, acesse o menu "File" e clique na opção "Open". Abra o arquivo que contém os componentes. Dê "Ok" e depois é só clicar em "Install".

• **Instaladores**: Com instaladores tudo se torna mais fácil, basta usar o instalador disponível e utiliza o "next-next-finish".

Fique por Dentro

Este artigo é útil para desenvolvedores e empresas que utilizam no Delphi componentes de terceiros, cuja instalação toma sempre algum tempo quando é necessário reinstalar ou iniciar um novo ambiente de desenvolvimento, devido à formatação das máquinas ou chegada de um novo membro na equipe. Desenvolveremos aqui um instalador de componentes que irá simplificar a adição destes adicionais ao IDE.

Todo esse procedimento realiza a instalação de apenas um pacote, então em cenários reais onde se tem diversos pacotes a serem instalados, todo o trabalho é multiplicado.

Independente da forma como são instalados, o Delphi utiliza uma única forma para localizar os componentes e exibi-los em suas devidas paletas, para isso ele utiliza o registro do Windows. Para definir os componentes a serem exibidos, o Delphi verifica se se existe um registro na seção "HKEY_CURRENT_USER\Software\Embarcadero\BDS\<Versão>\Known Packages", onde <Versão> seria a versão do Delphi que estamos utilizando. Além disso ele utiliza a seção "HKEY_CURRENT_USER\Software\Embarcadero\BDS\<Versão>\Palette\Cache" para identificar os componentes do pacote que serão apresentados na paleta. Dessa forma, para declarar um pacote e seus componentes no Delphi XE2, basta que declaremos a seção Known Packages, com isso, ao ler o método "Register" ele irá identificar seus componentes e registrá-los na seção Palette

Ao contrário de seu antecessor, o Delphi 7, no qual tínhamos que declarar os componentes manualmente, o XE2 possui recursos para facilitar essa operação. Para o correto funcionamento dos componentes devemos adicionar os caminhos utilizados na nossa Library Path e para isso utilizamos o

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

registro "HKEY_CURRENT_USER\Software\Embarcadero\BDS\<Versão>\Library\<Plataforma>\ Search Path". Aqui <plataforma> é a plataforma de compilação que o componente suporta, que pode ser OSX32, Win32, Win64 para Delphi XE2, e Android32, iOSDevice, IOSSimulator, OSX32, Win32, Win64 para o XE5.

Extensões de arquivos do Delphi

Para lidar corretamente com os diversos tipos de arquivos gerados pelo Delphi na criação de componentes, é preciso compreender para que serve cada uma das extensões utilizadas.

• PAS (Delphi Source File): Os arquivos PAS são sempre o código-fonte do programa e são os arquivos que contêm a maior parte do código em um aplicativo.

• DCU (Delphi Compiled Unit): É uma unidade PAS compilada. Por padrão, a versão compilada de cada unidade é armazenada em um arquivo no formato binário separado com o mesmo nome que o arquivo de unidade, mas com a extensão .DCU. Por exemplo, Unit1.dcu contém o código e os dados declarados no arquivo Unit1.pas. Quando você compila um projeto, as unidades individuais não são recompiladas, a menos que seu arquivo fonte (.pas) tenha sofrido alterações desde a última compilação, ou caso os arquivos .dcu não possam ser encontrados.

• DPK (Delphi Package): Este arquivo contém o código-fonte de um pacote, que na maioria das vezes é uma coleção de várias unidades. Eles são utilizados para construir bibliotecas de ligação dinâmica (DLLs) especiais chamadas pacotes (BPL).

• BPL (Borland Package Library): Este arquivo é como uma DLL do Windows, porém com recursos específicos do Delphi integrados a ela.

O que é o registro do Windows

Também conhecido como registro do sistema, Registro do Windows é um banco de dados hierárquico central no sistema operacional Windows, usado para armazenar as informações necessárias à configuração do sistema para um ou mais usuários, aplicativos e dispositivos de hardware.

O registro contém informações às quais o Windows faz referência continuamente durante a operação, como os perfis de cada usuário, os aplicativos instalados no computador e os tipos de documentos que cada um pode criar, configurações da folha de propriedades para ícones de pastas e aplicativos, o hardware existente no sistema e as portas que são usadas.

O registro substitui a maioria dos arquivos .ini com base em texto usados nos arquivos de configuração do Windows 3.x e do MS-DOS, como o Autoexec.bat e o Config.sys, assim concentra todas as informações editáveis do sistema operacional e da maioria dos softwares instalados, tornando a administração dessas informações mais fácil. Embora o registro seja comum para diversos sistemas operacionais Windows, existem algumas diferenças entre eles. Na **Figura 1** vemos o Editor do Registro (no Windows 10), que pode ser acessado através do comando "regedit" executado no prompt de comandos do Windows.

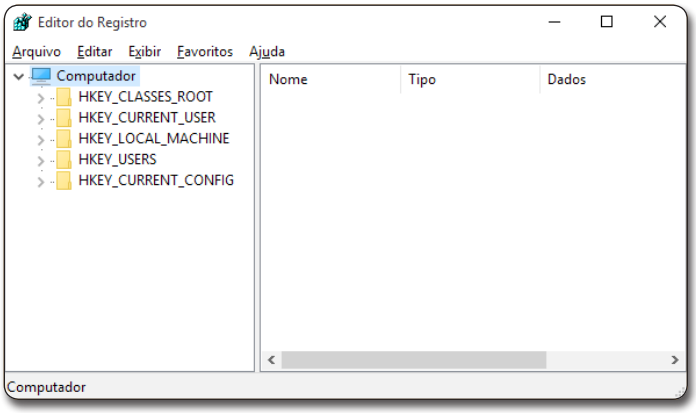


Figura 1. Editor do Registro do Windows

Uma ramificação do registro é um grupo de chaves, subchaves e valores que tem um conjunto de arquivos de suporte que contém o backup dos dados. Os arquivos de suporte para todas as ramificações, com exceção de HKEY_CURRENT_USER, estão na pasta %SystemRoot%\System32\Config no Windows NT 4.0, no Windows 2000, no Windows XP, no Windows Server 2003 e no Windows Vista. Os arquivos de suporte para HKEY_CURRENT_USER estão na pasta %SystemRoot%\Profiles\Username. As extensões do nome dos arquivos nestas pastas indicam o tipo de dados que elas contêm. Além disso, a falta de uma extensão pode às vezes indicar o tipo de dados que elas contêm.

Chaves de registro

Todas essas configurações do registro do Windows são editadas através de chaves, que são a unidade padrão de informação do registro. Por padrão, o Windows utiliza de 5 a 7 chaves principais que se subdividem em várias outras, de uma forma semelhante às pastas que estamos acostumados no Windows Explorer, cada uma referente à configuração de uma parte do sistema acompanhado de uma chave, como podemos observar na **Figura 1**. Cada uma dessas sub-chaves possui um valor e a mudança nessas sub-chaves é o que efetivamente realiza uma alteração. No Windows 7 e superiores, por exemplo, temos cinco chaves principais:

• **HKEY_CLASSES_ROOT (HKCR):** Presente nas versões atuais do Windows apenas para manter a compatibilidade com programas mais antigos, da geração 16 bits (dos tempos do DOS);

• **HKEY_CURRENT_USER (HKCU):** uma sub-chave de HKEY_USERS, contendo todas as configurações do usuário atualmente logado no sistema.

• **HKEY_LOCAL_MACHINE (HKLM):** Chave mais importante do registro, guarda todas as informações que o sistema operacional precisa para funcionar e de sua interface gráfica. Utiliza o arquivo SYSTEM para armazenar essas configurações;

• **HKEY_USERS (HKU):** Guarda as configurações de aparência do Windows e as configurações efetuadas pelos usuários, como papel de parede, protetor de tela, temas e outros, utilizando o arquivo USER para armazenar essas informações;

• **HKEY_CURRENT_CONFIG (HKCC):** Salva os perfis de hardware utilizados pelo usuário. Como normalmente só

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

é utilizado um perfil, o valor da chave é HKEY_LOCAL_MACHINECONFIG?00.

Na **Figura 2** vemos, no lado direito da tela, as chaves de uma das seções do registro. Dando dois cliques sobre uma chave, é possível alterá-la na tela que vemos na **Figura 3**.

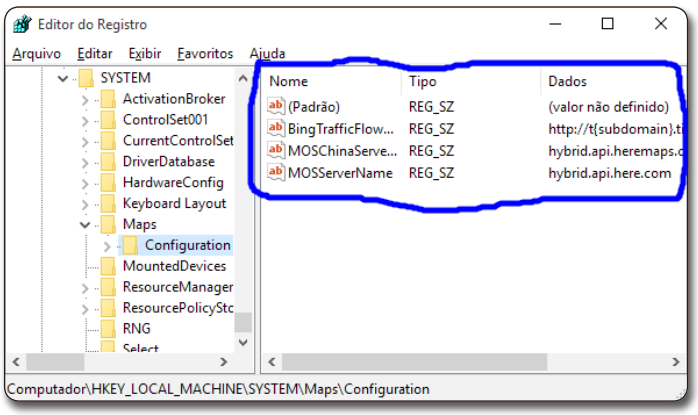


Figura 2. Chaves do registro

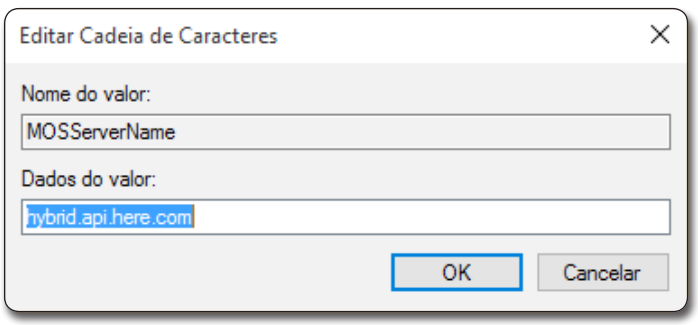


Figura 3. Alterando uma chave do registro

O registro pode ser utilizado para diversas finalidades, entre elas o controle de versão e licença de um software. Sendo necessário, é possível criar e alterar chaves no registro a partir do Delphi.

Os registros de componentes do Delphi

Para que possamos fazer um instalador, devemos conhecer quais registros devem ser utilizados e quais informações devem ser armazenadas nesses registros. Para o Delphi XE2, por exemplo, precisamos apenas trabalhar com dois registros:

• **Known Packages**: São as declarações dos pacotes que o Delphi irá utilizar. Com essa declaração estamos dizendo que ele conhece esses pacotes aqui citados e que eles devem ser disponibilizados para uso do programador, com isso o Delphi irá ler esses pacotes e extrair os componentes que estão registrados neles, exibindo-os em suas paletas.

• **Library**: É onde devemos informar onde os arquivos necessários para compilação se encontram e a qual plataforma esse arquivo corresponde. Para que o Delphi saiba dessas informações devemos declará-las na "Search Path" correspondente às plataformas que esse pacote suporta. Com isso indicamos para o Delphi que ele além de disponibilizar esses pacotes, quando houver uma referência a eles também informamos qual o caminho a ser utilizado para busca de suas DCU's e PAS's.

Conhecendo outros registros do Delphi

Vamos agora ver outros registros que podem ser úteis em outras ocasiões:

• **AutoRun**: Registro responsável por determinar os aplicativos que vão iniciar junto com o Delphi. Nele se encontra o UpdateCheck, sistema de atualização do Delphi. Caso tenha interesse em adicionar um novo aplicativo ou serviço para o Delphi, deverá ser utilizando esse registro;

• **Closed Files**: Contém uma lista de registro com os últimos arquivos abertos e o registro "Max Closed Files" é o que determina a quantidade de itens que ficaram listados;

• **Closed Projects**: Contém uma lista dos últimos projetos abertos, enquanto o registro "Max Closed Files" é o que determina a quantidade de itens que ficaram listados, assim como no Closed Files;

• **Code Explorer**: Contém uma lista das propriedades do Code Explorer, podemos habilitar ou desabilitar uma propriedade através desse registro;

• **Compiling**: Registro onde estão definidas as opções de compilação. Assim como no menu de opções do IDE, podemos habilitar ou desabilitar uma funcionalidade da compilação através desse registro. Por exemplo, podemos habilitar o fechamento automático do progresso definindo o "Auto Close Progress Dialog" como True;

• **Debugging**: Nesse registro podemos definir algumas configurações para o processo de debug dos projetos no IDE.

• **Editor**: Neste registro temos todas as configurações do nosso editor, uma série de definições podem ser feitas através dessa chave ou suas filhas;

• **Form Design**: Serve para definir funções dos forms, como auto criar os formulários, quanto à utilização de Grid no formulário, tamanho da Grid e apresentação dos nomes dos componentes;

• **Object Inspector**: Temos nesse registro uma série de definições para o Object Inspector, desde cores a serem definidas para as propriedades de um objeto ou a apresentação de uma opção;

• **Search**: Aqui é onde está determinando as configurações para a utilização do search. Podemos definir através desse registro a utilização de case sensitive, por exemplo, nas nossas pesquisas e até a utilização de expressões regulares.

Note que para garantir o armazenamento seguro de informações importantes, o Delphi utilizou o registro do Windows ao invés de arquivos locais, como os arquivos .ini comumente utilizados para armazenar configurações.

Contornando o Problema

Uma solução para contornar esse problema é o chamando "backup de Configuração", que consiste em exportar a seção Borland do registro do Windows e importar em uma na máquina desejada, mas isso tem algumas limitações. Uma delas é que

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

os componentes devem estar no mesmo diretório. Outro fato a ser levado em consideração é que não estará só restaurando os componentes, mas toda a configuração da antiga máquina, o que em alguns casos pode ser uma vantagem. Se limitarmos somente aos registros correspondentes aos componentes, ainda teríamos o problema dos diretórios que deveriam ser os mesmos. Isso seria uma solução, mas ainda não seria a melhor, então vamos à nossa solução ideal.

Criando o instalador

A partir de agora desenvolveremos no Delphi um instalador de componentes que fará a inserção destes no registro do Windows. Essa aplicação consistirá basicamente de três formulários, sendo o primeiro o de boas-vindas; no segundo será solicitada a pasta de distribuição dos arquivos necessários para o funcionamento dos componentes; e o terceiro formulário será o indicador de progresso da instalação. Uma proposta de interface para esses formulários pode ser vista nas **Figuras 4, 5** e **6,** respectivamente. Como este não é um fator determinante para o funcionamento da aplicação, fica a critério do leitor a definição da interface da forma que julgar mais adequado.

No primeiro formulário adicionaremos apenas uma descrição para o nosso instalador e dois botões, sendo um para cancelar a operação e outro para prosseguir. Esse formulário também será responsável por efetuar a chamada do nosso próximo formulário, que será o de seleção da pasta de instalação representado na **Figura 5.** Na **Listagem 1** podemos observar o código da unit main, que é o código do primeiro formulário.

Nesse código foram definidas duas variáveis globais: "PathInstall" e "PathExe" nas linhas 29 e 30, respectivamente. Para tratar o evento de clique do botão de Cancelar declaramos o método "btn_CancelarClick", onde implementamos apenas a finalização da aplicação com uma chamada ao método Application.Terminate na linha 41. Para o botão de Próximo declaramos o método "btn_ProximoClick", onde foram implementadas apenas duas linhas de código, sendo a linha 47 para interromper a exibição do formulário inicial e a 48 para apresentamos o formulário de diretórios (**Figura 5**) como modal. Para o evento Create do formulário temos o método "FormCreate", que pega o caminho de onde está sendo executada a aplicação e passa para a variável "PathExe" que foi definida na seção global.

Para o segundo formulário foram adicionados uma descrição e um Edit onde será informado o diretório de instalação. No rodapé foram adicionados mais dois botões, sendo um deles para retornar à tela anterior e outro para prosseguir com a instalação no diretório especificado, efetuando a chamada do próximo formulário, que indicará o progresso da instalação representado pela **Figura 6**. Na **Listagem 2** temos o código desse formulário.

Nessa unit temos os dois métodos que irão tratar o evento OnClick dos botões do rodapé. No método "Btn_ProximoClick" verificamos se o edit se encontra vazio e, em caso positivo, alertamos o usuário. Se o edit foi preenchido adequadamente, atribuímoso seu conteúdo à variável PathInstall e avançamos o processo de instalação exbindo o próximo formulário. O "Btn_VoltarClick" esconde o form atual e exibe o form principal novamente, retornando ao passo anterior da instalação.

Na unit Finalizar ficará concentrada toda a parte de inteligência da instalação dos componentes, o que não é um processo complicado.

Figura 4. Formulário principal do Instalador

Figura 5. Formulário de seleção de diretorio

Figura 6. Formulário de progresso da instalação

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Vamos ter uma lista de todos os pacotes que serão instalados e para cada pacote o seu devido caminho. Teremos uma lista de caminhos a serem adicionados no Libray Path, diretório onde o Delphi busca os arquivos dos componentes instalados. As listas serão preenchidas em um método separado, facilitando assim a manutenção do sistema, visto que a adição dos pacotes ficará concentrada em um único local e não espalhada pelo sistema. Isso nos permite futuramente remover esses itens do código e colocar em um arquivo .ini, por exemplo, ou ainda em outro mecanismo de armazenamento adequado.

Listagem 1. Código da unit main

```
unit Main;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
 Forms,Dialogs,lib, Menus, Vcl.StdCtrls, Vcl.Imaging.jpeg, Vcl.ExtCtrls;

type
 TFrm_Main = class(TForm)
  imgFundo: TImage;
  lbl2: TLabel;
  lbl1: TLabel;
  lblLblVer: TLabel;
  Panel1: TPanel;
  btn_Cancelar: TButton;
  btn_Proximo: TButton;
  procedure FormCreate(Sender: TObject);
  procedure btn_ProximoClick(Sender: TObject);
  procedure btn_CancelarClick(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Frm_Main : TFrm_Main;
 PathInstall  : String;
 PathExe    : String;

implementation

{$R *.dfm}

uses Path;

// Termina a aplicação
procedure TFrm_Main.btn_CancelarClick(Sender: TObject);
begin
 Application.Terminate;
end;

// Chama o próximo formulario
procedure TFrm_Main.btn_ProximoClick(Sender: TObject);
begin
 Self.Hide;
 Frm_Path.ShowModal;
end;

// Pega o diretorio do arquivo e define como diretorio de busca dos
// arquivos para distribuição
procedure TFrm_Main.FormCreate(Sender: TObject);
begin
 PathExe  := ExtractFileDir(Application.ExeName);
end;

end.
```

Listagem 2. Código da unit Path

```
unit Path;

interface

uses
 Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
 System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs,
 cxGraphics, Vcl.ExtCtrls, Vcl.StdCtrls, dxGDIPlusClasses,jpeg;

type
 TFrm_Path = class(TForm)
  img1: TImage;
  lbl1: TLabel;
  lbl2: TLabel;
  Edt_Pasta: TEdit;
  Panel1: TPanel;
  btn_Voltar: TButton;
  Btn_Proximo: TButton;
  imgFundo: TImage;
  BrowserDialog: TOpenDialog;
  procedure btn_VoltarClick(Sender: TObject);
  procedure Btn_ProximoClick(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;

var
 Frm_Path: TFrm_Path;

implementation
Uses Finalizar, Main;

{$R *.dfm}


procedure TFrm_Path.Btn_ProximoClick(Sender: TObject);
begin
 // Verifica se o caminho esta em branco
 if Trim(Edt_Pasta.Text) = '' then
 begin
  ShowMessage('O caminho de Instalação não pode ficar vazio. Verifique!');
  Exit;
 end;

 // Passa o valor capturado para a variavel PathInstall Global
 PathInstall := edt_Pasta.Text;

 Self.Hide;
 Frm_Finalizar.ShowModal;
end;

procedure TFrm_Path.btn_VoltarClick(Sender: TObject);
begin
 Self.Hide;
 Frm_Main.ShowModal;
end;

end.
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

O processo de instalação será dividido em duas etapas principais, cada uma com sub-etapas, porém todas de fácil compreensão. Para facilitar o entendimento, podemos observar um esquema desse funcionamento na **Figura 7**.

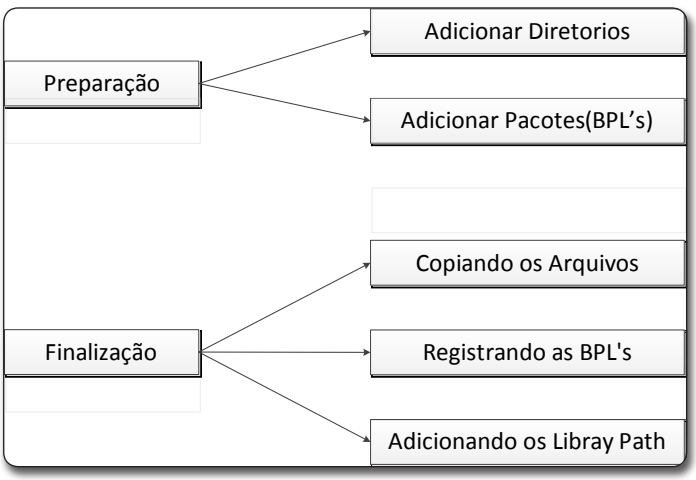


Figura 7. Fluxo das etapas de instalação.

A primeira etapa, chamada de Preparação, é onde será efetuada a chamada aos métodos que irão adicionar os pacotes e os diretórios as nossas listas. Já na etapa de finalização teremos três simples sub-etapas: copiar os arquivos de onde estão localizados (nesse caso os arquivos estão localizados na pasta do executável); em seguida iremos percorrer a lista de pacotes e efetuar a gravação no registro do Windows dentro da seção Known Packages do registro do Delphi; e por fim iremos percorrer e gravar os diretórios dos arquivos dos componentes no Library Path.

Para armazenar a lista de pacotes vamos adicionar em nosso formulário um componente TClientDataset contendo dois campos do tipo String, um para armazenar o nome do pacote e o segundo para armazenar o diretório onde esse pacote se encontra.

Para armazenar a listagem da Library Path vamos utilizar uma string que terá seu valor concatenado com ";" e adicionado ao registro Library do Delphi. Todo esse processo está devidamente modularizado em vários métodos que são descritos a seguir.

• **PrepararComponentes**: apresentado na **Listagem 3**, este é o método que usaremos para chamar os métodos de listar os diretórios e listar os pacotes;

• **ListarPath:** método no qual listaremos os diretórios que serão adicionados no Library Path e adicionaremos os caminhos na lista "SearchPath". Nesse método devem ser adicionados todos os caminhos necessários para o correto funcionamento dos componentes. O código desse método encontra-se na **Listagem 4**;

• **ListarPacotes:** nesse método informaremos todos pacotes que iremos instalar, adicionaremos os pacotes na lista para que possamos utilizá-los na segunda etapa. Seu código pode ser visto na **Listagem 5;**

• **AddPacote**: apresentado na **Listagem 6,** foi criado para não se replicar o código de insert na tabela de pacotes a cada novo pacote adicionado;

• **Finalizar**: nesse método, cujo código pode ser visto na **Listagem 7**, declaramos a variável Registry do tipo TRegistry e implementamos toda a cópia e registro dos componentes;

• **CopyAll:** esse método será utilizado para copiar todos os arquivos da pasta e subpastas onde está o executável para a pasta onde serão instalados os componentes. Nesse método utilizamos um ProgressBar para apresentar o progresso ao usuário, como podemos ver na **Listagem 8**;

• **CountFilesInFolder:** este método retorna o número de arquivos encontrados na pasta passada por parâmetro. Seu código encontra-se na **Listagem 9**;

• **FormActivate**: de acordo com a **Listagem 10**, ao ativar o formulário damos o "ponta pé" para nossa instalação. Nesse método são feitas as chamadas ao PrepararComponentes e ao Finalizar.

O método PrepararComponentes serve para organizarmos as tarefas iniciais da instalação, sendo responsável por efetuar a chamada aos métodos ListarPath e ListarPacotes (linhas 3 e 4), que farão a cópia e registro dos pacotes.

Listagem 3. Método PrepararComponentes

```
01 procedure TFrm_Finalizar.PrepararComponentes;
02 begin
03   ListarPath;
04   ListarPacotes;
05 end;
```

Listagem 4. Método ListarPath

```
01 procedure TFrm_Finalizar.ListarPath;
02 begin
03   //Deve ser montada a lista de todos os Caminhos a serem adicionados
04   SearchPath.Add('\Exemplo\Bin\');
05   SearchPath.Add('\Exemplo\Lib\');
06   SearchPath.Add('\Exemplo\Src\');
07 end;
```

No método ListarPath devemos adicionar todos os diretórios necessários para o funcionamento dos componentes na lista SearchPath. As pastas Bin, Lib e Src são geralmente utilizadas nas bibliotecas de componentes, não sendo, contudo, uma regra que todas elas precisem ser adicionadas à lista.

Nas primeiras linhas do método ListarPacotes preparamos o ClientDataset e em seguida adicionamos os pacotes, passando como primeiro argumento o nome do componente e como segundo a sua pasta.

O método AddPacote é responsável por evitar redundância no código no momento de adicionarmos novos ao DataSet.

O método Finalizar é dividido em três etapas, conforme foi esquematizado na **Figura 7**. Nele foram declaradas três variáveis: I, um contador para iterações; Registry, objeto utilizado para gravar e recuperar dados do registro do Windows; e Str_LibPath, uma string que será utilizada para receber a concatenação dos diretórios a serem inseridos nos Libray Path.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Nas linhas 8 e 9 instanciamos o objeto Registry e definimos que a chave raiz que representará é a HKEY_CURRENT_USER. Em seguida, nas linhas 11 e 12 alteramos o caption do Lbl_Install para apresentar o usuário que estamos iniciando o processo de instalação de componentes e iniciamos o procedimento.

No passo 1 de 3, que inicia na linha 15, alteramos o caption do Lbl_Install para interagir com o usuário e informar que vamos iniciar a cópia das pastas e seus conteúdos para a pasta de origem. Em seguida chamamos o método CopyAll, que é responsável por copiar todo o conteúdo da pasta origem para a pasta destino.

Na segunda etapa, que começa na linha 20, além da interação com usuário com o uso de label e progressbar, também percorremos a tabela de pacotes e para cada pacote é criado um item no registro do Windows através do método Registry.WriteString. Antes de se escrever em uma chave, porém, essa chave deve ter sido "aberta", e para isso utilizamos o método OpenKey da classe TRegistry (linha 27).

Para finalizar, na terceira etapa, que começa na linha 36, pegamos o valor atual do Library Path em seguida percorremos nossa lista de diretórios, concatenando o seu valor com o valor do registro atual para não perder as os dados já existentes. Após efetuar os loops necessários, substituímos o valor do registro atual pelo novo valor obtido através da concatenação utilizando também o Registry.WriteString. Por fim exibimos uma mensagem de conclusão para o usuário e fechamos a nossa aplicação, finalizando assim o processo de instalação dos componentes.

O método CopyAll, cujo código vemos na **Listagem 8,** recebe dois parâmetros: a pasta de origem e a pasta de destino. Esse método é responsável por percorrer o diretório de origem e copiar os arquivos para o diretório de destino. Trata-se de um método recursivo e a cada iteração, ao ser encontrado um subdiretório, o método é chamado novamente para a copiar do seu subdiretório de origem para o destino.

Nesse método utilizamos uma ProgressBar para apresentar ao usuário o andamento do processo. Primeiramente é criada a pasta destino e em seguida utilizamos o método FindFirst para saber se existe arquivos ou diretórios no diretório de origem. Então fazemos um loop com até não existirem mais itens nessa pasta. Na linha 20 é feita uma validação para ver se o arquivo não é uma pasta, se for uma pasta, então chamamos a função CopyAll, caso seja um arquivo, este é copiado para o destino. Na linha 10 dessa listagem utilizamos o método auxiliar CountFilesInFolder, que retorna a quantidade de arquivos existentes em um dado diretório recebido por parâmetro. Esse valor é utilizado para definir também o valor máximo da progressbar. O código desse método pode ser visto na **Listagem 9.**

Listagem 5. Método ListarPacotes

```
01 procedure TFrm_Finalizar.ListarPacotes;
02 begin
03   tb_pacotes.Close;
04   tb_pacotes.CreateDataSet;
05
06  //Adiciona as BPLs
07   AddPacote('DevMediaComps','Components\');
0o end;
```

Listagem 6. Método AddPacote

```
01 // Adicionar na tabela tb_pacotes as informações dos pacotes
02 procedure TFrm_Finalizar.AddPacote(Pacote, Path: String);
03 begin
04   tb_pacotes.Append;
05   tb_pacotes.FieldByName('Pacote'   ).AsString   := Pacote;
06   tb_pacotes.FieldByName('Path'     ).AsString   := Path;
07   tb_pacotes.Post;
08 end;
```

Listagem 7. Método Finalizar

```
01 procedure TFrm_Finalizar.Finalizar;
02 var I : Integer  ;
03    Registry   : TRegistry;
04    Str_LibPath : String   ;
05 begin
06
07   // Instancia um objeto de acesso ao registro do Windows
08   Registry := TRegistry.Create;
09   Registry.RootKey := HKEY_CURRENT_USER;
10
11   Lbl_Install.Caption := 'Preparando Instalação...';
12   Application.ProcessMessages;
13
14   // 1/3 Copia as Pastas para o local de instalação
15   Lbl_Install.Caption := '1/3 - Copiando Arquivos';
16   Application.ProcessMessages;
17   CopyAll(PathExe,PathInstall);
18
19   // 2/3 Registrar BPL's
20   Lbl_Install.Caption := '2/3 - Registrando BPL"s';
21   ProgressBar.Position := 0;
22   ProgressBar.Max     := tb_Pacotes.RecordCount;
23   tb_pacotes.First;
24   while Not(tb_pacotes.Eof) do
25   Begin
26    ProgressBar.Position := ProgressBar.Position +1;
27    Registry.OpenKey ('\Software\Embarcadero\BDS\9.0\
      Known Packages', True);
28    Registry.WriteString( PathInstall +
29                  tb_Pacotes.FieldByName('Path' ).AsString +
30                  tb_Pacotes.FieldByName('pacote').AsString + '.bpl',
31                  tb_Pacotes.FieldByName('pacote').AsString);
32                   tb_pacotes.Next;
33  end;
34
35  // 3/3 Adicionando os Libray Path
36  Lbl_Install.Caption := '3/3 - Adicionando os Libray Path';
37  Application.ProcessMessages;
38  If Registry.OpenKey('\Software\Embarcadero\BDS\9.0\Library\Win32',
    False)  Then
39     Str_LibPath := Registry.ReadString('Search Path');
40
41    For I := 0 to SearchPath.Count-1 do
42     Str_LibPath := Str_LibPath + ';' + PathInstall + SearchPath[i];
43
44    Registry.WriteString('Search Path',Str_LibPath);
45
46    ShowMessage('Instalação concluída.');
47    Application.Terminate;
48  end;
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 8. Método CopyAll

```
01 // Copia de forma recursiva todos os arquivos do destino
     // para a origem informada
02 Procedure TFrm_Finalizar.CopyAll(pOrigem,pDestino : String);
03 var
04  SR  : TSearchRec;
05 begin
06  //Cria a pasta de Origem
07  CreateDir(pDestino);
08
09  ProgressBar.Position      := 0;
10  ProgressBar.Max := CountFilesInFolder(pOrigem);
11
12  if FindFirst(pOrigem + '*.*', faAnyFile, SR) = 0 then
13  repeat
14   ProgressBar.Position := ProgressBar.Position +1;
15   Application.ProcessMessages;
16
17   if (SR.Name = '.') or (SR.Name = '..') then
18    Continue;
19
20   if DirectoryExists(pOrigem + SR.Name) then
21    CopyAll(pOrigem + SR.Name + '\',  pDestino + SR.Name + '\');
22
23   CopyFile(PChar(pOrigem + SR.Name), PChar(pDestino + SR.Name), true);
24
25  until FindNext(SR) <> 0;
26  FindClose(SR);
27
28 End;
```

Listagem 9. Método CountFilesInFolder

```
01 // Conta o Número de Registro em uma pasta
02 function TFrm_Finalizar.CountFilesInFolder(pFolder : String):Integer;
03 var
04  SR   : TSearchRec;
05  count : Integer;
06 begin
07  count := 0;
08  if FindFirst(pFolder + '*.*', faAnyFile, SR) = 0 then
09  begin
10   repeat
11    if (SR.Name = '.') or (SR.Name = '..') then
12     Continue;
13
14    Inc(count);
15   until FindNext(SR) <> 0;
16   FindClose(SR);
17  end;
18
19  Result := count;
20 end;
21
22 end.
```

Por fim, não podemos esquecer que os métodos de preparação e de finalização devem ser chamados em algum momento, e este momento é no evento OnActivate do formulário. O método onde tratamos esse evento está apresentado na **Listagem 10**. Com isso finalizamos o instalador e já podemos testá-lo.

Compilando componentes

Para que possamos utilizar os componentes no instalador, eles devem estar já compilados, por isso temos uma limitação na hora de trabalhar com diferentes versões do Delphi, pois caso tenhamos mais de uma versão, precisaremos ter as BPLs para cada uma delas. A solução para essa limitação é compilar os pacotes antes de instalar o componente, mas não faremos isso de forma manual, vamos apreender aqui a automatizar o processo de compilação.

Listagem 10. Método do Evento Activate do formulário de finalizar

```
01  procedure TFrm_Finalizar.FormActivate(Sender: TObject);
02  begin
03   PrepararComponentes;
04   Finalizar;
05 end;
```

Utilizaremos aqui o compilador do Delphi por linha de comando, que fica localizado na pasta "Bin" de sua instalação com o nome dcc32.exe. Ao executar o compilador devemos passar alguns parâmetros para definirmos o projeto a ser compilado, pasta de seus arquivos e onde devem ser gerados os arquivos binários. Para conhecer mais sobre os parâmetros do compilador, basta que digitemos dcc32 no prompt de comando do Windows e os detalhes serão apresentados, conforme é descrito a seguir

- **-B:** Indica que serão compiladas todas as units do projeto, é equivalente ao Build do IDE;
- **-Q:** Executa uma compilação silenciosa;
- **-H:** Habilita ou desabilita a exibição de hints, para isso utilizamos na sequência o sinal de + ou - para habilitar ou desabilitar;
- **-W:** Habilita ou desabilita a exibição dos Warnings e assim como nos hints, utilizamos os sinais de + ou - para habilitar ou desabilitar;
- **-R:** Conjunto de diretórios onde o compilador irá buscar por arquivos de resources (.RC, .RES) utilizados no projeto;
- **-U:** Conjunto de diretórios onde o compilador irá procurar os arquivos .PAS ou .DCU utilizados no projeto;
- **-E:** Define diretório de output para DLL e EXE.

O Delphi disponibiliza um compilador diferente para cada plataforma, ou seja, se seu projeto for para 64 bit ou para uma plataforma mobile, deverá ser utilizado outro compilador, de acordo com as especificações a seguir.

- **DCC32.EXE:** Compilador para 32-bit;
- **DCC64.EXE:** Compilador para 64-bit;
- **DCCOSX.EXE:** Compilador OS X;
- **DCCIOS32.EXE:** Compilador para iOS Simulator;
- **DCCIOSARM.EXE:** Compilador para dispositivos iOS;
- **DCCAARM.EXE:** Compilador para Android.

Vamos então compilar nosso primeiro projeto. Para isso devemos criar um novo projeto, depois abrir o prompt de comando e digitarmos a seguinte linha de comando: *"dcc32 <Caminho_**do**_Projeto>"*, onde *"<Caminho_do_Projeto>"* é o diretório do projeto mais o nome do dpr. Ao executar esse comando será retornado um erro, indicando que o arquivo ".dcu" do projeto não foi encontrado. Esse erro ocorre porque nenhum ".pas" nem ".dcu" com os nomes correspondentes ao projeto foram encontrados nos diretórios

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

conhecidos pelo dcc32. Uma forma simples de evitar esse problema é sempre iniciar a compilação a partir do diretório do projeto. Para isso, basta alterar o diretório corrente antes de executar o compilador. Se tudo correr bem, a compilação será executada sem erros, e o Meu_Projeto.exe será gerado.

Com isso já conseguimos criar uma forma de automatizar a compilação dos nossos componentes, então vamos criar um outro projeto onde vamos listar as versões do Delphi instaladas na máquina e em seguida compilar a BPL com o compilador especifico da versão. Poderíamos efetuar essa implementação direto no instalador, em uma etapa antes da instalação, pois é o momento em que vamos gerar as BPL's para que o instalador as utilize. Nesse exemplo a aplicação será criada separada para que não tenhamos um excesso de informação em um projeto.

Nosso projeto será composto de um formulário para seleção da versão, e uma classe, que executará a compilação. Vamos então criar um formulário, que deve ficar semelhante à **Figura 8**, porém as versões serão definidas em tempo de execução, bastando adicionar na tela um TRadioGroup.

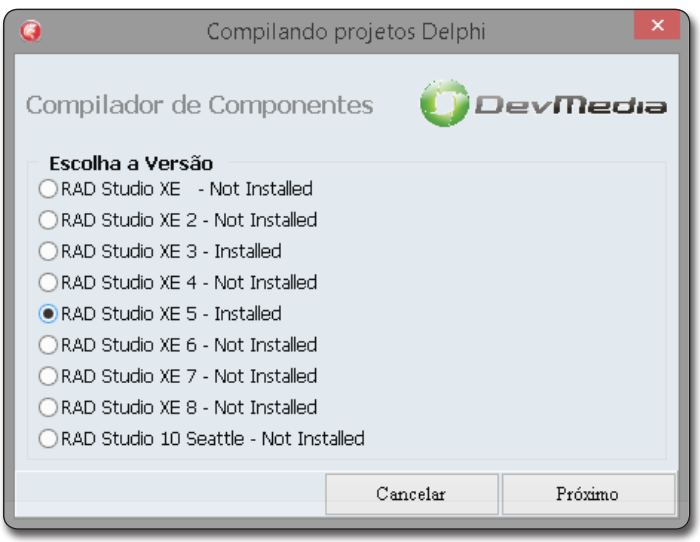


Figura 8. Tela do compilador de projetos.

Na unit principal criaremos uma constante que será utilizada para listar as versões instaladas do Delphi, a partir da XE, de acordo com a **Listagem 11.**

Definimos aqui um array constante de nove elementos, cada um contendo um número de versão e o nome comercial da mesma. O primeiro campo será usado para comparar com as versões instaladas no Windows, enquanto a segunda parte será exibida para o usuário. Na **Listagem 12** temos o método que será responsável por retornar o caminho de instalação de uma versão especifica, que deve ser declarado no escopo privado da unit.

Nesse método estamos trabalhando com um "TRegistry" para ler o registro "RootDir" da pasta "Versao". Essa chave contém o valor do caminho de instalação do Delphi. Caso a versão não esteja instalada, o retorno será vazio, caso contrário teremos o caminho de instalação para que possamos chamar o compilador correspondente à versão desejada.

Em seguida, na **Listagem 13** implementamos o método que irá nos listar as versões do Delphi para que o usuário decida para qual delas deseja instalar os componentes.

Listagem 11. Array de versões

```
const
 Versoes : array[0..8]  of
                array[0..1] of string =
        (('8.0','RAD Studio XE '),
        ('9.0','RAD Studio XE 2'),
        ('10.0','RAD Studio XE 3'),
        ('11.0','RAD Studio XE 4'),
        ('12.0','RAD Studio XE 5'),
        ('13.0','RAD Studio XE 6'),
        ('14.0','RAD Studio XE 7'),
        ('15.0','RAD Studio XE 8'),
        ('17.0','RAD Studio 10 Seattle')
      );
```

Listagem 12. Retornar caminho de instalação do Delphi

```
function TMain.RetornaRegistro(Versao : String):string;
var
 Registro : TRegistry;
begin
  Registro := TRegistry.Create(KEY_READ);
  Registro.RootKey:=HKEY_CURRENT_USER;

  result := '';
  if registro.OpenKey('\Software\Embarcadero\BDS\'+ Versao, False) then
   result := Registro.ReadString('RootDir');

  registro.CloseKey;
  registro.Free;
end;
```

Listagem 13. Listando versões do Delphi

```
procedure TMain.ListaVersoes;
Var
  i          : Integer;
  sNmVer     : String;
  sPathVer   : String;
  nVerInstalled : Integer;
begin
  rg_Versoes.Items.Clear;
  for I := 0 to High(Versoes) do
  Begin
    sNmVer := Versoes[i][1];

    sPathVer := RetornaRegistro(Versoes[i][0]);
    if (sPathVer.IsEmpty) then
     begin
      sNmVer := sNmVer + ' - Not Installed';
     end
    else
     begin
       sNmVer := sNmVer + ' - Installed';
       nVerInstalled := i;
     end;

    rg_Versoes.Items.Add(sNmVer);
  End;

  rg_Versoes.ItemIndex := nVerInstalled;
end;
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

O algoritmo desse método é bem simples, efetuamos um loop sobre nosso array de versões e para cada iteração verificamos e armazenamos o caminho da instalação da versão contida na posição zero do segundo array para uma variável chamada "sPathVer", essa etapa se encontra na linha 13. Na linha 14 validamos se a variável está vazia e se isso ocorrer é por que aquela versão não está instalada na máquina, então a variável "sNmVer" recebe a descrição da versão mais uma descrição de não instalado. Caso a variável "sPathVer" esteja com valor, então temos a versão instalada e adicionamos à descrição uma informação que a versão está instalada. Na linha 21 passamos para a variável "nVerInstalled" o valor do índice, isso é para que ao final deixemos já selecionada a última versão do Delphi encontrada. Para cada iteração, na linha 24 é adicionado um item ao "rg_Versoes" com o valor da "sNmVer". Com isso listamos as versões para o usuário e descriminamos se a versão está ou não instalada.

Nosso próximo passo é criar um método que irá retornar o caminho da versão selecionada pelo usuário, conforme mostra a **Listagem 14.**

O método tem como resultado o retorno da função "RetornaRegistro", que recebe como parâmetro a posição no array correspondente ao item selecionado no TRadioGroup.

Na sequência, temos na **Listagem 15** o método Compilar, que é o real responsável por executar os comandos que vimos previamente para os componentes selecionados.

Listagem 14. Pega caminho da versão selecionada

```
01 function TMain.GetCompiladorSelecionado: String;
02 begin
03   Result := RetornaRegistro(Versoes[rg_Versoes.ItemIndex][0]) + 'bin\dcc32.exe';
04 end;
```

Listagem 15. Metodo compilar

```
01 procedure TMain.Compilar;
02 Const
03   sPathPrj = '<Meu_Caminho>\MinhaBPL.dpk';
04 Var
05   Compilador : TCompilador;
06 begin
07   Compilador := TCompilador.Create;
08   try
09    Compilador.Compilador := GetCompiladorSelecionado;
10
11    Compilador.Listpaths.Add(ExtractFilePath(sPathPrj));
12
13    Compilador.ProjetPath := sPathPrj;
14    Compilador.OutputPath := gsAppPath;
15    Compilador.Executar;
16  finally
17    FreeAndNil(Compilador);
18  end;
19 end;
```

Nesse método declaramos e instanciamos um objeto do tipo TCompilador, cuja definição veremos mais adiante. Essa classe terá como propriedade o caminho do projeto, o caminho do output e a lista de diretórios. No método "Compilar" instanciamos e preenchemos os valores dessas propriedades. No caso da propriedade "Listpaths", ela é uma TStringList à qual devem ser adicionados todos os diretórios necessários para a compilação do pacote. O método "Executar" da classe TCompilador é responsável por pegar esses valores passados para suas propriedades e montar o arquivo bat de compilação.

Com isso, praticamente concluímos nosso formulário principal, basta que no botão de cancelar executemos um comando Close ou Application.Terminate e no botão de "Próximo/Instalar" adicionemos a chamada ao método Compilar definido na **Listagem 15**. Agora vejamos na **Listagem 16** a definição da classe TCompilador, que deve preferencialmente ser criada em uma unit separada.

Listagem 16. Escopo da classe TCompilador

```
01  TCompilador = class
02    private
03      fParametros : TStringList;
04      fListpaths  : TStringList;
05      fProjetPath : String;
06      fOutputPath : String;
07      fCompilador : String;
08
09      procedure prepararCompilacao;
10    public
11      Constructor Create;
12      Destructor Destroy; override;
13
14      Procedure Executar;
15    published
16      property Compilador : String read fCompilador write fCompilador;
17      property ProjetPath : String read fProjetPath write fProjetPath;
18      property OutputPath : String read fOutputPath write fOutputPath;
19      property Listpaths  : TStringList read fListpaths;
20
21  end;
```

Os fields fParametros e fListpaths devem ser instanciados no Create e liberadas no Destroy. Para o "prepararCompilacao" será implementado o código da **Listagem 17** e para o "Executar", o código da **Listagem 18**.

Montamos em tempo de execução um bat definindo o projeto, o diretório do Delphi e os diretórios do projetos e diretórios de output. Os parâmetros "LE","LN" e NU definem os outputs para cada tipo de arquivo:

- **-LE:** Define o output para os arquivos .bpl;
- **-LN:** Define o output para os arquivos .dcp;
- **-NU:** Define o output para os arquivos .dcu.

Após a criação do arquivo, o salvamos no caminho do executável com um nome definido em uma constante chamada "sNameComand" definida no topo de nossa unit com o valor "Command.cmd".

Na linha 4 efetuamos uma chamada ao método prepararCompilacao, que nos monta o arquivo bat para execução, em seguida na linha 6 passamos o retorno da chamada a função ShellExecute, passando como parâmetro o nome do arquivo bat criado.

Se o valor de RetExec for maior que 32 significa que a execução desse script foi efetuada e então mostramos uma mensagem

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

indicando a compilação bem-sucedida, caso seja menor, a função ShellExecute teve algum problema na execução do mesmo e caso isso aconteça é emitida uma mensagem informando o código do erro.

Listagem 17. Método prepararCompilacao

```
01 procedure TCompilador.prepararCompilacao;
02 Var
03  i    : integer;
04  sPath : String;
05 begin
06  fParametros.Clear;
07  fParametros.Add('set PRJ=' + ProjetPath +'');
08  fParametros.Add('set DIR_PRJ='+ ExtractFilePath(ProjetPath)+'');
09  fParametros.Add('set DIR_DELPHI=""+ Compilador+"");
10  fParametros.Add('set DIR_OUTPUT='+ OutputPath);
11
12  fParametros.Add('cd %DIR_PRJ%');
13  fParametros.Add('%DIR_DELPHI% -B %PRJ% -E%DIR_OUTPUT%
    -NU%DIR_OUTPUT% - LE%DIR_OUTPUT% -LN%DIR_OUTPUT% -Q');
14
15  fParametros.SaveToFile(gsAppPath + sNameCommand);
16 end;
```

Listagem 18. TCompilador.Executar

```
01 procedure TCompilador.Executar;
02 Var RetExec : Integer;
03 begin
04  prepararCompilacao;
05
06  RetExec := ShellExecute(Application.Handle, 'open', char(sNameCommand),nil,
    nil, SW_HIDE);
07
08  if (RetExec >  32) then
09   ShowMessage('Projeto compilado...')
10  else
11   ShowMessage('Erro ' + RetExec.ToString + ' ao compilar o projeto.')
12 end;
```

Para esse projeto ainda poderíamos efetuar alguns incrementos, como adicionar uma checklist para definirmos os componentes que devem ser instalados, carregar os arquivos de uma pasta na rede, carregar os nomes e pasta dos componentes do banco de dados, para que os componentes não sejam reinstalados sem necessidade, validando quais os componentes já estão instalados e carregando somente os não instalados. Poderíamos ainda verificar se alguns dos componentes, mesmo que instalados, não estão desatualizados ou faltando algum arquivo e efetuar esses ajustes automaticamente. De forma complementar, poderíamos também criar um atualizador e chamá-lo na execução do Delphi, ou ainda como serviço, para validar a necessidade de atualizações dos componentes.

Devemos observar as necessidades e tentar, sempre que possível, automatizá-las, garantindo assim ganho de produtividade, uma vez que o tempo antes utilizado para tarefas repetitivas, poderia então ser dedicado a atividades mais práticas de desenvolvimento.

Podemos utilizar o recurso do registro do Windows também para outras finalidades, como definir ou alterar padrões no ambiente do Delphi, adicionar menus, verificar quais os últimos projetos abertos por um determinado usuário, adicionar ou remover funcionalidades no Code Explorer, definir padrões do editor, entre outras coisas. Podemos alterar praticamente tudo a partir do registro do Windows, pois quase todas as configurações estão definidas lá, não só do Delphi como também de muitos outros softwares, até o próprio sistema operacional.

Autor

Gutierry Antonio
gutierrydsn@hotmail.com
Software Engineer \ MySQL DBA. Cursando especialização em Engenharia de Sistemas, Graduado em Sistemas da informação e Análise de Sistemas na Universidade Estácio de Sá. Experiência em desenvolvimento com Visual Fox Pro, Delphi, Ruby, Action Script 3.0, Java, C# e C++ e bancos de dados Firebird, MySQL e SQL Server.

Links:

Extensões de arquivos do Delphi
http://delphi.about.com/od/beginners/a/aa032800a.htm

Informações do Registro do Windows para usuários avançados
https://support.microsoft.com/pt-br/kb/256986/pt-br

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Conhecimento faz diferença!

Faça já sua assinatura digital ! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional.
Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software.
Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviçoes); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DEVMEDIA

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única.
Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486